g1-finalized-main-coding

September 14, 2024

1 1.0 Data Understanding

[ ]:
```
import pandas as pd
import numpy as np
```

col variable get from "anomaly network detection.docx"

all the text occur inside the col variable is representing the heading for the "networkTrain.txt" data

"'

1 Duration: Length of time duration of the connection 2 Protocol_type: Protocol used in the connection 3 Service: Destination network service used 4 Flag: Status of the connection – Normal or Error 5 Src_bytes: Number of data bytes transferred from source to destination in single connection 6 Dst_bytes: Number of data bytes transferred from destination to source in single connection 7 Land: if source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0 8 Wrong_fragment: Total number of wrong fragments in this connection 9 Urgent: Number of urgent packets in this connection. Urgent packets are packets with the urgent bit Activated

CONTENT RELATED FEATURES OF EACH NETWORK CONNECTION VECTOR 10 Hot: Number of „hot indicators in the content such as: entering a system directory, creating programs and executing programs 11 Num_failed _logins: Count of failed login attempts 12 Logged_in Login Status: 1 if successfully logged in; 0 otherwise 13 Num_compromised: Number of `compromised' ' conditions 14 Root_shell: 1 if root shell is obtained; 0 otherwise 15 Su_attempted: 1 if`su root" command attempted or used; 0 otherwise 16 Num_root: Number of `root'' accesses or number of operations performed as a root in the connection 17 Num_file_creations: Number of file creation operations in the connection 18 Num_shells: Number of shell prompts 19 Num_access_files: Number of operations on access control files 20 Num_outbound_cmds: Number of outbound commands in an ftp session 21 Is_hot_login: 1 if the login belongs to the`hot" list i.e., root or admin; else 0 22 Is_guest_login: 1 if the login is a "guest" login; 0 otherwise

TIME RELATED TRAFFIC FEATURES OF EACH NETWORK CONNECTION VECTOR 23 Count: Number of connections to the same destination host as the current connection in the past two seconds 24 Srv_count: Number of connections to the same service (port number) as the current connection in th e past two seconds 25 Serror_rate: The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in count (23) 26 Srv_serror_rate: The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in srv_count (24) 27 Rerror_rate: The percentage of con-

nections that have activated the flag (4) REJ, among the connections aggregated in count (23) 28 Srv_rerror_rate: The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in srv_count (24) 29 Same_srv_rate: The percentage of connections that were to the same service, among the connections aggregated in count (23) 30 Diff_srv_rate: The percentage of connections that were to different services, among the connections aggregated in count (23) 31 Srv_diff_host_ rate: The percentage of connections that were to different destination machines among the connections aggregated in srv_count (24) HOST BASED TRAFFIC FEATURES IN A NETWORK CONNECTION VECTOR 32 Dst_host_count: Number of connections having the same destination host IP address 33 Dst_host_srv_ count: Number of connections having the same port number 34 Dst_host_same *srv_rate: The percentage of connections that were to the same service, among the connections aggregated in dst_host_count (32) 35 Dst_host_diff* srv_rate: The percentage of connections that were to different services, among the connections aggregated in dst_host_count (32) 36 Dst_host_same *src_port_rate: The percentage of connections that were to the same source port, among the connections aggregated in dst_host_srv_c ount (33) 37 Dst_host_srv* diff_host_rate: The percentage of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count (33) 38 Dst_host_serro r_rate: The percentage of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_count (32) 39 Dst_host_srv_s error_rate: The percent of connections that have activated the flag (4) s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_c ount (33) 40 Dst_host_rerro r_rate: The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_count (32) 41 Dst_host_srv_r error_rate: The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst_host_srv_c ount (33)

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
col =␣
 ↪["duration","protocol_type","service","flag","src_bytes","dst_bytes","land",
"wrong_fragment","urgent","hot","num_failed_logins","logged_in",
"num_compromised","root_shell","su_attempted","num_root","num_file_creations",
"num_shells","num_access_files","num_outbound_cmds","is_host_login",
"is_guest_login","count","srv_count","serror_rate", "srv_serror_rate",
"rerror_rate","srv_rerror_rate","same_srv_rate", "diff_srv_rate",␣
 ↪"srv_diff_host_rate","dst_host_count","dst_host_srv_count","dst_host_same_srv_rate",
"dst_host_diff_srv_rate","dst_host_same_src_port_rate",
"dst_host_srv_diff_host_rate","dst_host_serror_rate","dst_host_srv_serror_rate",
"dst_host_rerror_rate","dst_host_srv_rerror_rate","attack", "last_flag"]
```

[ ]:
```
df = pd.read_csv("/content/drive/MyDrive/Machine Learning /Assignment/
 ↪NetworkTrain.txt", sep=',', names = col)
```

[ ]:
```
df.head()
```

[ ]:
```
df.to_csv("networkDetect.csv", index = False)
```

```
[ ]: data = pd.read_csv("/content/drive/MyDrive/Machine Learning /Assignment/
      ↪networkDetect.csv")
     data.head()
```

```
[ ]: data.shape
```

```
[ ]: data.dtypes
```

```
[ ]: data.describe().T
```

```
[ ]: data.select_dtypes(exclude=[np.number])
```

2 2.0 Data Preprocessing / Data Cleaning

2.1 Missing value : use to identify and handle for missing value 2.2 Identify and remove duplicate value 2.3 Data encoding : convert categorical data to integer value 2.4 dropping unwanted data and combine data

2.1 2.1 Handling Missing Value

handling missing value is one of the useful technique used in preprocessing. handling with missing value could enhance the model performance and remove with unwanted value

```
[ ]: missingData = data.isnull()
     missingData
```

the isnull() function has showing false result of all column which indicate that all of the column appear in the dataset has no null value

```
[ ]: data.info()
```

using another technique of info() to double the null value occcur in the dataset but there is no null value occur in the dataset

```
[ ]: missing_counts = data.isnull().sum()
     missing_counts
```

Lastly, using of isnull().sum() function to find total of missing value exists in the dataset. The result above has showing that 0 of null value in the dataset.

```
[ ]: import missingno as msno

     msno.matrix(data)
```

Summary of 2.1 Handling Missing Value By using several technique of identify missing value, there is a continuous result that indicate that there is no missing value occur in the dataset. So the dataset can be consider that there is no issue in missing value and doesn't need perform

any handling missing value method such as replacing the value or removing unwanted data. Since there is no any issue, the dataset will proceed to 2.2 outlier detection

2.2 2.2 identify and remove duplicate value

Idenfity ad remove duplicate value could benefit user on further training their data into forecast model and this also enhance the accuracy of a model

[ ]:
```
duplicate = data.duplicated().sum()
data = data.drop_duplicates()

print("Duplicate data in dataset: " + str(duplicate))
```

to check duplicate data inside the data but the result has shown that there is no duplicate data in the dataset

2.3 2.3 Data encoding

The process of data labeling is provide more meaningful insight on a unqiue value that contains in the column

[ ]:
```
# protocol_type service flag attack
print("protocol type element : ")
print(data['protocol_type'].unique())

print("\n")
print("service type element : ")
print(data['service'].unique())

print("\n")
print("flag type element : ")
print(data['flag'].unique())

print("\n")
print("attack type element : ")
print(data['attack'].unique())
```

checking the unique value in the categorical variable

[ ]:
```
print("protocol type element : ")
print(data['protocol_type'].value_counts())

print("\n")
print("service type element : ")
print(data['service'].value_counts())

print("\n")
print("flag type element : ")
print(data['flag'].value_counts())
```

```
print("\n")
print("attack type element : ")
print(data['attack'].value_counts())
```

checking every value in the categorical variable

[ ]:
```
# Assuming 'data' is your DataFrame
top_service = data['service'].value_counts().head().index
all_service = np.array(data['service'])

# Replace less frequent services with 'others'
all_service = [service if service in top_service
               else 'others' for service in all_service]

# Convert to DataFrame
all_service_df = pd.DataFrame(all_service, columns=['service'])


# If you want to integrate this back into your original DataFrame
data['service'] = all_service_df

# Display the updated DataFrame
data['service'].unique()
```

getting the top 5 value in the service

[ ]:
```
top_flag = data['flag'].value_counts().head(5).index
all_flag = np.array(data['flag'])

# Replace less frequent services with 'others'
all_flag = [flag if flag in top_flag
               else 'others' for flag in all_flag]

# Convert to DataFrame
all_flag_df = pd.DataFrame(all_flag, columns=['flag'])


# If you want to integrate this back into your original DataFrame
data['flag'] = all_flag_df

# Display the updated DataFrame
data['flag'].unique()
```

[ ]:
```
top_attack = data['attack'].value_counts().head(5).index
all_attack = np.array(data['attack'])

# Replace less frequent services with 'others'
```

```
all_attack = [attack if attack in top_attack
              else 'others' for attack in all_attack]

# Convert to DataFrame
all_attack_df = pd.DataFrame(all_attack, columns=['attack'])


# If you want to integrate this back into your original DataFrame
data['attack'] = all_attack_df

# Display the updated DataFrame
data['attack'].unique()
```

[ ]:
```
categorical_cols = data.select_dtypes(include=['object']).columns
categorical_cols
```

[ ]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

encoder = OneHotEncoder(sparse_output=False)
```

[ ]:
```
oneHotEncoded = encoder.fit_transform(data[categorical_cols])

oneHotEncoded
```

[ ]:
```
oneHotDf = pd.DataFrame(oneHotEncoded,
                        columns=encoder.get_feature_names_out(categorical_cols))

oneHotDf
```

[ ]:
```
df_encoded = pd.concat([data,oneHotDf], axis=1)
df_encoded = df_encoded.drop(categorical_cols,axis=1)
```

2.4 2.4 dropping unwanted data and combine data

[ ]:
```
dataDrop = df_encoded
dataDrop.head()
```

[ ]:
```
dataDrop['total_bytes'] = dataDrop['src_bytes'] + dataDrop['dst_bytes']
```

[ ]:
```
dataDrop.drop(columns=['src_bytes','dst_bytes'],axis=1,inplace=True)
dataDrop.drop(columns=['is_host_login','num_failed_logins'],axis=1,inplace=True)
```

```
[ ]: dataDrop.to_csv("/content/drive/MyDrive/Machine Learning /Assignment/
      ↪NetworkPreprocess.csv", index = False)
```

after the preprocess, save the preprocess part to csv file

3 3.0 Exploratory Data Analysis

Exploratory Data Analysis (EDA) act as a crucial step in machine learning project due to EDA provide the overview of data and the characteristics of data in a dataset. Usually EDA involve with visualization technique which user are able to review the data physically through various technique such as histogram and heatmap.

3.1 Feature Importance 3.2 Bar plot and Histogram 3.3 Dimensionality Reduction 3.4 Correlation 3.5 Factor Analysis 3.6 Pairplot 3.7 Treemaps

3.1 3.1 Feature Importance

Feature Importance is a process of calculating the score of feature before implement with modeling. The highest score it gains from the calculation, representing that the feature is more important for the target variable.

Reference: https://builtin.com/data-science/feature-importance#:~:text=Feature%20importance%20refers%20to

import related python library

```
[ ]: from sklearn.ensemble import RandomForestClassifier
     from sklearn.preprocessing import StandardScaler
     from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt
     import seaborn as sns
     import pandas as pd
```

```
[ ]: data_clean = pd.read_csv("/content/drive/MyDrive/Machine Learning /Assignment/
      ↪NetworkPreprocess.csv")
     data_clean.head()
```

```
[ ]: attack_columns = [col for col in data_clean.columns if 'attack_' in col]
     print("Attack Columns " + str(attack_columns))
```

This code creates a list called attack_columns containing all column names from the data_clean DataFrame that include the substring 'attack_', and then prints this list with the label "Attack Columns".

```
[ ]: # attack normal
     target_variable = 'attack_normal'
     x = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
      ↪col])
     y = data_clean[target_variable]
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x, y)

feature_importances = pd.DataFrame(rf.feature_importances_, index=x.columns,␣
 ↪columns=['importance']).sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(13, 14))
sns.barplot(x='importance', y=feature_importances.index,␣
 ↪data=feature_importances)
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Features / attributes')
plt.show()
```

This code trains a Random Forest classifier to predict the attack_normal target variable using features from the data_clean DataFrame (excluding columns with 'attack_' in their names), then calculates and plots the feature importances to visualize which features are most influential in the model's predictions, with the plot displaying the importance of each feature on the y-axis and their corresponding importance scores on the x-axis. As we can see for attack_normal ,src_bytes in having the highest posibility.

[ ]:
```
# attack ipsweep
target_variable = 'attack_ipsweep'
x = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean[target_variable]

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x, y)

feature_importances = pd.DataFrame(rf.feature_importances_, index=x.columns,␣
 ↪columns=['importance']).sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(13, 14))
sns.barplot(x='importance', y=feature_importances.index,␣
 ↪data=feature_importances)
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Features / attributes')
plt.show()
```

[ ]:
```
# attack neptune
target_variable = 'attack_neptune'
x = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
```

```
y = data_clean[target_variable]

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x, y)

feature_importances = pd.DataFrame(rf.feature_importances_, index=x.columns,␣
 ↪columns=['importance']).sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(13, 14))
sns.barplot(x='importance', y=feature_importances.index,␣
 ↪data=feature_importances)
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Features / attributes')
plt.show()
```

[ ]:
```
# attack portsweep
target_variable = 'attack_portsweep'
x = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean[target_variable]

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x, y)

feature_importances = pd.DataFrame(rf.feature_importances_, index=x.columns,␣
 ↪columns=['importance']).sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(13, 14))
sns.barplot(x='importance', y=feature_importances.index,␣
 ↪data=feature_importances)
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Features / attributes')
plt.show()
```

[ ]:
```
# attack satan
target_variable = 'attack_satan'
x = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean[target_variable]

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x, y)
```

```
feature_importances = pd.DataFrame(rf.feature_importances_, index=x.columns,␣
 ↪columns=['importance']).sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(13, 14))
sns.barplot(x='importance', y=feature_importances.index,␣
 ↪data=feature_importances)
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Features / attributes')
plt.show()
```

[ ]:
```
# attack others
target_variable = 'attack_others'
x = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean[target_variable]

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(x, y)

feature_importances = pd.DataFrame(rf.feature_importances_, index=x.columns,␣
 ↪columns=['importance']).sort_values('importance', ascending=False)

# Plot feature importances
plt.figure(figsize=(13, 14))
sns.barplot(x='importance', y=feature_importances.index,␣
 ↪data=feature_importances)
plt.title('Feature Importance from Random Forest')
plt.xlabel('Importance')
plt.ylabel('Features / attributes')
plt.show()
```

3.2 3.2 bar plot and histogram

[ ]:
```
# Bar plots for one-hot encoded categorical features
categorical_cols = [col for col in data_clean.columns if 'protocol_type_' in␣
 ↪col or 'service_' in col or 'flag_' in col or 'attack_' in col]

plt.figure(figsize=(12, 8))
data_clean[categorical_cols].sum().sort_values().plot(kind='barh')
plt.title('Bar Plot of Categorical Features')
plt.show()
```

This code identifies columns in the data_clean DataFrame that contain the substrings ‘proto-

col_type_', 'service_', 'flag_', or 'attack_' and stores them in a list called categorical_cols. It then creates a horizontal bar plot to visualize the sum of values in these categorical columns, with the plot sorted by the sum of each feature, helping to highlight the distribution and prevalence of these categorical features in the dataset.

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Identify the categorical columns related to one-hot encoded features
categorical_cols = [col for col in data_clean.columns if 'protocol_type_' in␣
 ↪col or 'service_' in col or 'flag_' in col or 'attack_' in col]

# Select only numerical columns by excluding categorical columns
numerical_cols = [col for col in data_clean.columns if col not in␣
 ↪categorical_cols]

# Plot histograms for numerical features
data_clean[numerical_cols].hist(bins=30, figsize=(25, 20))
plt.suptitle('Histograms of Numerical Features')
plt.show()
```

This code generates histograms for all numerical features in the data_clean DataFrame, using 30 bins for each histogram. The histograms are displayed in a grid format with a large figure size (25x20), providing a visual overview of the distribution of each numerical feature in the dataset. The plot is titled "Histograms of Numerical Features" to summarize the content of the visualization.

3.3 3.3 Dimesionality Reduction using principle component analysis (PCA)

Principal Components: PCA transforms the original high-dimensional data into a new set of orthogonal (uncorrelated) components called principal components. These components are linear combinations of the original features and are ordered by the amount of variance they capture from the data.

Variance Explained: The first principal component captures the most variance in the data, the second captures the second most variance, and so on. By choosing n_components, you decide how many of these principal components you want to retain in your analysis. reference: - https://machinelearningmastery.com/principal-component-analysis-for-visualization/

- https://builtin.com/data-science/step-step-explanation-principal-component-analysis#:~:text=Principal%20component%20analysis%2C%20or%20PCA,information%20in%20the%20larg

Standard Scaler Formula Before applying PCA, the data is usually centered by subtracting the mean of each feature (often done automatically if using a StandardScaler). This centering means that the transformed data can have positive or negative values depending on how the original features relate to each principal component.

import relevant library

[ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
attack_columns = [
    'attack_normal',
    'attack_ipsweep',
    'attack_neptune',
    'attack_portsweep',
    'attack_satan',
    'attack_others'
]

# Drop the target variables from the features
X = data_clean.drop(columns=attack_columns)

# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in X.columns if X[col].nunique() == 2]

# Separate continuous columns
continuous_cols = [col for col in X.columns if col not in binary_cols]

# Apply StandardScaler to the continuous data
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = X[binary_cols].reset_index(drop=True)

# Combine the target variables into a single categorical variable
# We assume that each row can only belong to one attack category
y_combined = pd.Series(['None'] * len(data_clean), index=data_clean.index)

for attack in attack_columns:
    y_combined[data_clean[attack] == 1] = attack

# Perform PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_prepared)

# Define a custom color palette
palette = {
    'attack_normal': 'red',
    'attack_ipsweep': 'blue',
    'attack_neptune': 'green',
    'attack_portsweep': 'purple',
```

```
    'attack_satan': 'orange',
    'attack_others': 'brown',
    'None': 'black'
}

# Visualize the PCA components
plt.figure(figsize=(10, 8))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y_combined, palette=palette,␣
 ↪alpha=0.5)
plt.title('2D PCA of Network Data')
plt.xlabel('attribute')
plt.ylabel('attack types')
plt.show()
```

This code preprocesses the data_clean DataFrame for Principal Component Analysis (PCA) by first separating the target variable attack_normal (stored in y) and dropping any columns related to attacks from the feature set X. It then identifies binary columns (those containing only 0s and 1s) and separates the continuous columns. The continuous data is standardized using StandardScaler, and the resulting scaled data is combined with the binary columns to form X_prepared. PCA is applied to reduce the dimensionality of the data to 2 components, which are then visualized using a scatter plot, with points colored based on the target variable y to illustrate how the data separates in the reduced 2D space. The plot is titled "2D PCA of Network Data," with axes labeled to indicate the features and attack/normal classification.

3.4 3.4 Correlation

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
corr_matrix = data_clean.corr()

# Generate and display the correlation heatmap for the dataset
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix, annot=False, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix Heatmap')
plt.show()
```

This code calculates the correlation matrix of a cleaned dataset (data_clean) to measure the linear relationships between variables, and then generates and displays a heatmap of this matrix using Matplotlib and Seaborn. The heatmap visually represents the strength and direction of correlations, with colors indicating positive or negative relationships between variables, making it easier to identify patterns or insights in the data.

[ ]:
```
import pandas as pd

correlation_matrix = data_clean.corr()
```

```
# Convert to DataFrame for easier manipulation
corr_data = correlation_matrix.stack().reset_index()
corr_data.columns = ['feature_1', 'feature_2', 'correlation']

# Filter based on absolute value and unique pairs
filtered_corr = corr_data[
    (abs(corr_data['correlation']) >= 0.6) &
    (corr_data['feature_1'] != corr_data['feature_2'])
].sort_values(by='correlation', ascending=False)

# Display the filtered correlations
filtered_corr
```

This code calculates the correlation matrix of the cleaned dataset (data_clean) and then converts it into a more manipulable DataFrame format (corr_data). It renames the columns for clarity, identifying the correlated feature pairs and their correlation values. The code then filters out correlations with an absolute value less than 0.6 and removes duplicate or self-correlated pairs (i.e., where a feature is correlated with itself). Finally, it sorts the filtered correlations in descending order and displays the resulting pairs that have strong correlations, making it easier to identify significant relationships between features.

##3.5 Factor analysis

Factor Analysis (FA) is a statistical technique primarily used to identify and model the underlying relationships between observed variables. Its main purposes include:

Dimensionality Reduction: - Like PCA, factor analysis can reduce the number of variables in a dataset while retaining the essential information. However, it does so by focusing on identifying latent (hidden) factors that explain the correlations between observed variables.

Identifying Latent Constructs: - Factor analysis helps in identifying latent variables (also known as factors) that are not directly observed but are inferred from the correlations among observed variables. For example, in psychology, latent constructs like intelligence or personality traits might be inferred from a set of observed behaviors or test scores. reference: - https://www.upgrad.com/us/blog/what-is-factor-analysis-and-its-types/#:~:text=Factor%20analysis%20is%20an%20unsupervised,data%20analysis%20much%20more%20efficient.

[ ]:
```
import pandas as pd
from sklearn.decomposition import FactorAnalysis
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in data_clean.columns if data_clean[col].nunique()␣
 ↪== 2]

# Separate continuous columns
```

```
continuous_cols = [col for col in data_clean.columns if col not in binary_cols]

# Assuming 'attack_normal' is the target variable
X = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean['attack_normal']

# Apply StandardScaler to the continuous data
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = data_clean[binary_cols].reset_index(drop=True)

# Apply Factor Analysis
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X_prepared)

# Visualize the Factor Analysis components
plt.figure(figsize=(10, 8))
colors = ['red', 'blue']  # Assuming binary classification (0 or 1)
plt.scatter(X_fa[:, 0], X_fa[:, 1], c=y, cmap=plt.cm.coolwarm, alpha=0.5)
plt.title('Factor Analysis Components')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.colorbar(label='Class')
plt.show()
```

This code preprocesses the data_clean DataFrame for Factor Analysis by first identifying binary columns (those with exactly 2 unique values) and separating continuous columns. It then drops columns related to attacks from the feature set and standardizes the continuous data using StandardScaler. The standardized continuous data is combined with the binary columns to form X_prepared. Factor Analysis is performed to reduce the data to 2 components, which are visualized in a scatter plot with points colored based on the target variable attack_normal. The plot, titled "Factor Analysis Components," shows how the data points are distributed in the reduced feature space, with the colorbar indicating the class labels (0 or 1).

[ ]:
```
# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in data_clean.columns if data_clean[col].nunique()␣
 ↪== 2]

# Separate continuous columns
continuous_cols = [col for col in data_clean.columns if col not in binary_cols]

# Assuming 'attack_normal' is the target variable
```

```
X = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean['attack_neptune']

# Apply StandardScaler to the continuous data
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = data_clean[binary_cols].reset_index(drop=True)

# Apply Factor Analysis
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X_prepared)

# Visualize the Factor Analysis components
plt.figure(figsize=(10, 8))
colors = ['red', 'blue']  # Assuming binary classification (0 or 1)
plt.scatter(X_fa[:, 0], X_fa[:, 1], c=y, cmap=plt.cm.coolwarm, alpha=0.5)
plt.title('Factor Analysis Components')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.colorbar(label='Class')
plt.show()
```

[ ]:
```
# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in data_clean.columns if data_clean[col].nunique()␣
 ↪== 2]

# Separate continuous columns
continuous_cols = [col for col in data_clean.columns if col not in binary_cols]

# Assuming 'attack_normal' is the target variable
X = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean['attack_ipsweep']

# Apply StandardScaler to the continuous data
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = data_clean[binary_cols].reset_index(drop=True)

# Apply Factor Analysis
```

```
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X_prepared)

# Visualize the Factor Analysis components
plt.figure(figsize=(10, 8))
colors = ['red', 'blue']  # Assuming binary classification (0 or 1)
plt.scatter(X_fa[:, 0], X_fa[:, 1], c=y, cmap=plt.cm.coolwarm, alpha=0.5)
plt.title('Factor Analysis Components')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.colorbar(label='Class')
plt.show()
```

[ ]:
```
# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in data_clean.columns if data_clean[col].nunique()␣
 ↪== 2]

# Separate continuous columns
continuous_cols = [col for col in data_clean.columns if col not in binary_cols]

# Assuming 'attack_normal' is the target variable
X = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean['attack_others']

# Apply StandardScaler to the continuous data
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = data_clean[binary_cols].reset_index(drop=True)

# Apply Factor Analysis
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X_prepared)

# Visualize the Factor Analysis components
plt.figure(figsize=(10, 8))
colors = ['red', 'blue']  # Assuming binary classification (0 or 1)
plt.scatter(X_fa[:, 0], X_fa[:, 1], c=y, cmap=plt.cm.coolwarm, alpha=0.5)
plt.title('Factor Analysis Components')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.colorbar(label='Class')
plt.show()
```

[ ]:
```
# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in data_clean.columns if data_clean[col].nunique()␣
 ↪== 2]

# Separate continuous columns
continuous_cols = [col for col in data_clean.columns if col not in binary_cols]

# Assuming 'attack_normal' is the target variable
X = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean['attack_portsweep']

# Apply StandardScaler to the continuous data
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = data_clean[binary_cols].reset_index(drop=True)

# Apply Factor Analysis
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X_prepared)

# Visualize the Factor Analysis components
plt.figure(figsize=(10, 8))
colors = ['red', 'blue']  # Assuming binary classification (0 or 1)
plt.scatter(X_fa[:, 0], X_fa[:, 1], c=y, cmap=plt.cm.coolwarm, alpha=0.5)
plt.title('Factor Analysis Components')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.colorbar(label='Class')
plt.show()
```

[ ]:
```
# Identify binary columns (those that contain only 0s and 1s)
binary_cols = [col for col in data_clean.columns if data_clean[col].nunique()␣
 ↪== 2]

# Separate continuous columns
continuous_cols = [col for col in data_clean.columns if col not in binary_cols]

# Assuming 'attack_normal' is the target variable
X = data_clean.drop(columns=[col for col in data_clean.columns if 'attack_' in␣
 ↪col])
y = data_clean['attack_satan']

# Apply StandardScaler to the continuous data
```

```
scaler = StandardScaler()
X_continuous_scaled = scaler.fit_transform(X[continuous_cols])

# Combine scaled continuous data with binary data
X_prepared = pd.DataFrame(X_continuous_scaled, columns=continuous_cols)
X_prepared[binary_cols] = data_clean[binary_cols].reset_index(drop=True)

# Apply Factor Analysis
fa = FactorAnalysis(n_components=2)
X_fa = fa.fit_transform(X_prepared)

# Visualize the Factor Analysis components
plt.figure(figsize=(10, 8))
colors = ['red', 'blue']  # Assuming binary classification (0 or 1)
plt.scatter(X_fa[:, 0], X_fa[:, 1], c=y, cmap=plt.cm.coolwarm, alpha=0.5)
plt.title('Factor Analysis Components')
plt.xlabel('Factor 1')
plt.ylabel('Factor 2')
plt.colorbar(label='Class')
plt.show()
```

3.5 3.6 Pair plot

Pair plot is a matrix graphs that use to visulize the relationship between each pair of variable in a dataset

[ ]:
```
data_clean.info()
```

[ ]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[ ]:
```
# Specify the target variable
y = data_clean['attack_normal']

# Combine the target variable with the features you want to plot
# Select a subset of continuous columns (for simplicity)
cols_to_plot = ['total_bytes', 'flag_SF', 'dst_host_same_srv_rate',␣
 ↪'last_flag','attack_normal']

# Create a DataFrame with only the selected columns
data_to_plot = data_clean[cols_to_plot]

# Create a pair plot
sns.pairplot(data_to_plot, hue='attack_normal', palette='coolwarm')

# Show the plot
```

```
plt.show()
```

This code creates a pair plot to visualize relationships between a subset of continuous and categorical features in the data_clean DataFrame, including src_bytes, dst_bytes, flag_SF, dst_host_same_srv_rate, last_flag, and the target variable attack_normal. The sns.pairplot function generates scatter plots for each pair of features and histograms for individual features, with points colored based on the attack_normal classification. The plot uses the coolwarm palette to distinguish between different classes, providing a comprehensive view of how feature pairs relate to each other and the target variable.

[ ]:
```
# Specify the target variable
y = data_clean['attack_ipsweep']

# Combine the target variable with the features you want to plot
# Select a subset of continuous columns (for simplicity)
cols_to_plot = ['dst_host_same_srv_rate', 'dst_host_count',␣
 ↪'protocol_type_icmp', 'dst_host_same_src_port_rate',␣
 ↪'total_bytes','attack_ipsweep']

# Create a DataFrame with only the selected columns
data_to_plot = data_clean[cols_to_plot]

# Create a pair plot
sns.pairplot(data_to_plot, hue='attack_ipsweep', palette='coolwarm')

# Show the plot
plt.show()
```

[ ]:
```
# Specify the target variable
y = data_clean['attack_neptune']

# Combine the target variable with the features you want to plot
# Select a subset of continuous columns (for simplicity)
cols_to_plot = ['same_srv_rate', 'dst_host_srv_serror_rate', 'total_bytes',␣
 ↪'serror_rate', 'flag_S0','attack_neptune']

# Create a DataFrame with only the selected columns
data_to_plot = data_clean[cols_to_plot]

# Create a pair plot
sns.pairplot(data_to_plot, hue='attack_neptune', palette='coolwarm')

# Show the plot
plt.show()
```

[ ]:
```
# Specify the target variable
y = data_clean['attack_satan']
```

```
# Combine the target variable with the features you want to plot
# Select a subset of continuous columns (for simplicity)
cols_to_plot = ['diff_srv_rate', 'dst_host_diff_srv_rate', 'total_bytes',␣
 ↪'count', 'same_srv_rate','attack_satan']

# Create a DataFrame with only the selected columns
data_to_plot = data_clean[cols_to_plot]

# Create a pair plot
sns.pairplot(data_to_plot, hue='attack_satan', palette='coolwarm')

# Show the plot
plt.show()
```

[ ]:
```
# Specify the target variable
y = data_clean['attack_portsweep']

# Combine the target variable with the features you want to plot
# Select a subset of continuous columns (for simplicity)
cols_to_plot = ['flag_RSTR', 'rerror_rate', 'dst_host_same_src_port_rate',␣
 ↪'dst_host_diff_srv_rate', 'dst_host_same_srv_rate','attack_portsweep']

# Create a DataFrame with only the selected columns
data_to_plot = data_clean[cols_to_plot]

# Create a pair plot
sns.pairplot(data_to_plot, hue='attack_portsweep', palette='coolwarm')

# Show the plot
plt.show()
```

[ ]:
```
# Specify the target variable
y = data_clean['attack_others']

# Combine the target variable with the features you want to plot
# Select a subset of continuous columns (for simplicity)
cols_to_plot = ['last_flag', 'total_bytes', 'srv_count', 'protocol_type_icmp',␣
 ↪'dst_host_srv_diff_host_rate','attack_others']

# Create a DataFrame with only the selected columns
data_to_plot = data_clean[cols_to_plot]

# Create a pair plot
sns.pairplot(data_to_plot, hue='attack_others', palette='coolwarm')

# Show the plot
```

```
plt.show()
```

3.6 3.7 Treemaps

Utlize of treemaps is because treemap provide a clear and consise way to visualize hierachical, proportional and categorical data. They allow for a deep understanding of the relationships and distribution of various features within the dataset, which can be critical in tasks like identifying attacks patterns or understanding network behavior.

Use of treemaps 1. Hierarchical Structure: - Use Case: dataset contains categorical features like attack_type, protocol_type, service, and flag, which often have a hierarchical or grouped nature. - Treemap Utility: Treemaps are excellent for visualizing hierarchical data, allowing user to see the structure and relative proportions of different categories within your dataset. 2. Visualizing Proportions: - Use Case: If user want to understand the distribution of different attack types, protocols, or services in your dataset, a treemap allows you to see how much each category contributes to the whole. - Treemap Utility: The size of each rectangle in the treemap corresponds to the relative frequency or magnitude of that category, making it easy to compare and understand the dataset's composition. Reference: https://python-graph-gallery.com/treemap/#:~:text=A%20treemap%20displays%20hierarchical%20data,rectangle%20positions%20and'

[ ]:
```
pip install squarify
```

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import squarify

# Count the occurrences of each attack type
attack_counts = data_clean[[col for col in data_clean.columns if 'attack_' in␣
 ↪col]].sum()

# Modify the labels to include the count
labels = [f'{label}\n({value})' for label, value in zip(attack_counts.index,␣
 ↪attack_counts)]

# Plot the treemap
plt.figure(figsize=(12, 8))
squarify.plot(sizes=attack_counts, label=labels, alpha=0.7)
plt.title('Treemap of Attack Types')
plt.axis('off')
plt.show()
```

4 4.0 Sampling method

Sampling methods are techniques used to select a subset of data from a larger dataset. This subset can then be analyzed to make inferences or to work with more manageable data sizes. The choice of sampling method depends on the data's characteristics and the analysis goals. For your dataset,

which deals with network intrusion detection, sampling methods could be used to balance the dataset, reduce data size, or improve model performance.

Reference: https://medium.com/@ayanchowdhury00/sampling-techniques-and-its-implementation-in-python-487995b9399c

[ ]:
```
import pandas as pd
dataSP = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪NetworkPreprocess.csv')
```

4.1 4.1 Random Sampling

[ ]:
```
dataSP.shape
```

[ ]:
```
def random_sampling(dataSP):
    return dataSP.sample(frac=0.1, random_state=42)

    # Perform random sampling
random_sampled_data = random_sampling(dataSP)

    # Display the first few rows of the sampled data
print("Random Sample:")
print(random_sampled_data)
```

##4.2 Systematic Sampling

[ ]:
```
def systematic_sampling(dataSP):
    total_rows = len(dataSP)
    sample_size = int(total_rows * 0.1)
    step_size = total_rows // sample_size
    return dataSP.iloc[::step_size, :]

    # Perform systematic sampling
systematic_sample = systematic_sampling(dataSP)

    # Display the first few rows of the sampled data
print("Systematic Sample:")
print(systematic_sample)
```

4.2 4.3 Incremental Sampling

[ ]:
```
def incremental_sampling(dataSP):
    desired_sample_size = int(len(dataSP) * 0.1)
    total_data_size = len(dataSP)
    batch_size = int(np.ceil(total_data_size / 2))  # Divide the data into 2␣
 ↪batches
    incremental_data = pd.DataFrame()
```

```
    for start_idx in range(0, total_data_size, batch_size):
        end_idx = min(start_idx + batch_size, total_data_size)
        batch_data = dataSP.iloc[start_idx:end_idx]
        sampled_batch = batch_data.sample(frac=desired_sample_size /␣
↪total_data_size, random_state=42)  # Sample fraction
        incremental_data = pd.concat([incremental_data, sampled_batch])

        if len(incremental_data) >= desired_sample_size:
            break

    if len(incremental_data) > desired_sample_size:
        incremental_data = incremental_data.sample(n=desired_sample_size,␣
↪random_state=42)

    return incremental_data

    # Perform incremental sampling
incremental_data = incremental_sampling(dataSP)

    # Display the first few rows of the sampled data
print("Incremental Sample:")
print(incremental_data)
```

4.3 4.4 Clustering Sampling

[ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import random

def cluster_sampling(dataSP):
    scaler = StandardScaler()
    data_normalized = scaler.fit_transform(dataSP)

    kmeans = KMeans(n_clusters=5, random_state=42)
    dataSP['cluster'] = kmeans.fit_predict(data_normalized)

    cluster_size = 2
    num_clusters = len(dataSP) // cluster_size
    n_clusters_to_sample = int(0.1 * num_clusters)

    clusters = [range(i * cluster_size, (i + 1) * cluster_size) for i in␣
↪range(num_clusters)]
    selected_clusters = random.sample(clusters, n_clusters_to_sample)
```

```
    selected_indices = [index for cluster in selected_clusters for index in␣
 ↪cluster]
    selected_indices = [i for i in selected_indices if i < len(dataSP)]

    return dataSP.iloc[selected_indices]

    # Perform clustering sampling
cluster_sample = cluster_sampling(dataSP)

    # Display the first few rows of the sampled data
print("Cluster Sample:")
print(cluster_sample)
```

4.4 4.5 Sampling method evaluation

###4.5.1 sample size of each sampling method

[ ]:
```
print("Random sampling     : ",len(random_sampled_data))
print("Incremental sampling: ",len(incremental_data))
print("Systematic sampling : ",len(systematic_sample))
print("Cluster sampling    : ",len(cluster_sample))
```

4.4.1 4.5.2 Memory usage comparison

Reference: https://www.analyticsvidhya.com/blog/2024/06/memory-profiling-in-python/#:~:text=install%20memory%2Dprofiler-,Usage,with%20the%20%2Dm%20memory_profiler%20Flag.&te

[ ]:
```
pip install memory_profiler
```

[ ]:
```
from memory_profiler import memory_usage

# Function to measure memory usage
def measure_memory_usage(func, *args):
    mem_usage = memory_usage((func, args))
    return max(mem_usage) - min(mem_usage)

# Example usage assuming dataSP is already loaded
dataSP = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪NetworkPreprocess.csv')  # Load your dataset here

# Measure memory usage for each method
mem_usage_random = measure_memory_usage(random_sampling, dataSP)
mem_usage_systematic = measure_memory_usage(systematic_sampling, dataSP)
mem_usage_incremental = measure_memory_usage(incremental_sampling, dataSP)
mem_usage_cluster = measure_memory_usage(cluster_sampling, dataSP)

print(f"Memory usage for Random Sampling: {mem_usage_random:.2f} MiB")
```

```
print(f"Memory usage for Systematic Sampling: {mem_usage_systematic:.2f} MiB")
print(f"Memory usage for Incremental Sampling: {mem_usage_incremental:.2f} MiB")
print(f"Memory usage for Cluster Sampling: {mem_usage_cluster:.2f} MiB")
```

###4.5.3 Execution time comparison

[ ]:
```
import time

def measure_execution_time(func, *args):
    start_time = time.time()
    result = func(*args)
    end_time = time.time()
    return end_time - start_time, result

# Example usage:
exec_time_random, _ = measure_execution_time(random_sampling, dataSP)
exec_time_systematic, _ = measure_execution_time(systematic_sampling, dataSP)
exec_time_incremental, _ = measure_execution_time(incremental_sampling, dataSP)
exec_time_cluster, _ = measure_execution_time(cluster_sampling, dataSP)

print(f"Execution Time for Random Sampling: {exec_time_random:.4f} seconds")
print(f"Execution Time for Systematic Sampling: {exec_time_systematic:.4f}␣
 ↪seconds")
print(f"Execution Time for Incremental Sampling: {exec_time_incremental:.4f}␣
 ↪seconds")
print(f"Execution Time for Cluster Sampling: {exec_time_cluster:.4f} seconds")
```

###4.5.4 Sampling accuracy

[ ]:
```
def compare_statistics(full_data, sampled_data):
    comparison = pd.DataFrame({
        'Full Data Mean': full_data.mean(),
        'Sampled Data Mean': sampled_data.mean(),
        'Difference': full_data.mean() - sampled_data.mean()
    })
    return comparison

# Compare statistics for each method:
_, random_sample = measure_execution_time(random_sampling, dataSP)
_, systematic_sample = measure_execution_time(systematic_sampling, dataSP)
_, incremental_sample = measure_execution_time(incremental_sampling, dataSP)
_, cluster_sample = measure_execution_time(cluster_sampling, dataSP)

print("Random Sampling Statistics Comparison:")
print(compare_statistics(dataSP, random_sample))

print("Systematic Sampling Statistics Comparison:")
```

```
print(compare_statistics(dataSP, systematic_sample))

print("Incremental Sampling Statistics Comparison:")
print(compare_statistics(dataSP, incremental_sample))

print("Cluster Sampling Statistics Comparison:")
print(compare_statistics(dataSP, cluster_sample))
```

###4.5.5 Bias and representativeness

[ ]:
```
import seaborn as sns
import matplotlib.pyplot as plt

def plot_distribution_comparison(full_data, sampled_data, column_name):
    plt.figure(figsize=(10, 6))
    sns.histplot(full_data[column_name], color='blue', label='Full Data',␣
 ↪kde=True)
    sns.histplot(sampled_data[column_name], color='red', label='Sampled Data',␣
 ↪kde=True)
    plt.legend()
    plt.title(f"Distribution of {column_name} - Full Data vs Sampled Data")
    plt.show()

# Compare distributions for a specific column (e.g., 'column_name') for each␣
 ↪method:
plot_distribution_comparison(dataSP, random_sample, 'attack_ipsweep')
plot_distribution_comparison(dataSP, systematic_sample, 'attack_ipsweep')
plot_distribution_comparison(dataSP, incremental_sample, 'attack_ipsweep')
plot_distribution_comparison(dataSP, cluster_sample, 'attack_ipsweep')
```

###4.5.6 Scalability

[ ]:
```
def measure_scalability(func, data, fractions=[0.1, 0.2, 0.5, 1.0]):
    results = []
    for frac in fractions:
        subset = data.sample(frac=frac, random_state=42)
        exec_time, _ = measure_execution_time(func, subset)
        mem_usage = measure_memory_usage(func, subset)
        results.append((frac, exec_time, mem_usage))
    return results

# Measure scalability for each method:
scalability_random = measure_scalability(random_sampling, dataSP)
scalability_systematic = measure_scalability(systematic_sampling, dataSP)
scalability_incremental = measure_scalability(incremental_sampling, dataSP)
scalability_cluster = measure_scalability(cluster_sampling, dataSP)
```

```
# Print or plot scalability results
print("Scalability Results (Fraction, Execution Time, Memory Usage):")
print(f"Random Sampling: {scalability_random}")
print(f"Systematic Sampling: {scalability_systematic}")
print(f"Incremental Sampling: {scalability_incremental}")
print(f"Cluster Sampling: {scalability_cluster}")
```

4.5 4.6 Summary of Comparison

Systematic Sampling emerges as the best sampling method based on these results, for the following reasons:

- Execution Time: It has the fastest execution time across all sample fractions, even improving as the sample fraction increases. This makes it highly efficient for processing large datasets.
- Memory Usage: Like random and incremental sampling, systematic sampling has minimal memory usage, but it outperforms the others in speed.
- Scalability: It scales exceptionally well, handling even the full dataset with virtually no increase in resource consumption.

Why Systematic Sampling is Best in This Case:

- Efficiency: The ability to sample very quickly makes systematic sampling ideal when time is a critical factor.
- Memory: The minimal memory footprint ensures that systematic sampling can be used on systems with limited resources or very large datasets without running into memory issues.
- Applicability: While systematic sampling can introduce bias if there's an underlying pattern in the data that matches the sampling interval, it is often a non-issue, especially when the data is randomly ordered or lacks such periodicity.

5 5.0 Preparation before clustering

5.1 5.1 standardize data

[ ]:
```
import pandas as pd
data = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
  ↪NetworkPreprocess.csv')
```

[ ]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Assuming 'data' is your original DataFrame

columns_to_standardize = [
    'duration', 'count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes',
```

```
    'serror_rate', 'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',
    'diff_srv_rate', 'srv_diff_host_rate', 'dst_host_same_srv_rate',
    'dst_host_diff_srv_rate', 'dst_host_same_src_port_rate',
    'dst_host_srv_diff_host_rate', 'dst_host_serror_rate',
    'dst_host_srv_serror_rate', 'dst_host_rerror_rate',
    'dst_host_srv_rerror_rate'
]

# Select the data to be standardized
data_to_standardize = data[columns_to_standardize]

# Apply StandardScaler
scaler_standard = StandardScaler()
data_standard_scaled = scaler_standard.fit_transform(data_to_standardize)
data_standard_scaled_df = pd.DataFrame(data_standard_scaled,␣
 ↪columns=columns_to_standardize)

# Apply MinMaxScaler
scaler_minmax = MinMaxScaler()
data_minmax_scaled = scaler_minmax.fit_transform(data_to_standardize)
data_minmax_scaled_df = pd.DataFrame(data_minmax_scaled,␣
 ↪columns=columns_to_standardize)

# Create two separate DataFrames: one for Standard Scaled data and one for␣
 ↪MinMax Scaled data
data_standard_scaled_full = data.copy()
data_minmax_scaled_full = data.copy()

# Replace the original columns with the Standard Scaled data in one DataFrame
data_standard_scaled_full[columns_to_standardize] = data_standard_scaled_df

# Replace the original columns with the MinMax Scaled data in the other␣
 ↪DataFrame
data_minmax_scaled_full[columns_to_standardize] = data_minmax_scaled_df

# Save to CSV if needed
standard_scaled_csv = '/content/drive/MyDrive/Machine Learning /Assignment/
 ↪StandardScaler.csv'
minmax_scaled_csv = '/content/drive/MyDrive/Machine Learning /Assignment/
 ↪MinMaxScaling.csv'

data_standard_scaled_full.to_csv(standard_scaled_csv, index=False)
data_minmax_scaled_full.to_csv(minmax_scaled_csv, index=False)
```

5.2 5.2 Utilize systematic sampling to reduce the size of data

[ ]:
```
import pandas as pd

dataMinMax = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪MinMaxScaling.csv')
dataScaler = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪StandardScaler.csv')

print(len(dataMinMax))
print(len(dataScaler))
```

[ ]:
```
def systematic_sampling(dataSP):
    total_rows = len(dataSP)
    sample_size = int(total_rows * 0.07)
    step_size = total_rows // sample_size
    return dataSP.iloc[::step_size, :]

    # Perform systematic sampling
minmax_sampling = systematic_sampling(dataMinMax)
standard_sampling = systematic_sampling(dataScaler)

    # Display the first few rows of the sampled data
print("Systematic Sample (min max):")
print(len(minmax_sampling))
print("\n\nSystematic Sample (standard scaler):")
print(len(standard_sampling))
```

[ ]:
```
minmax_sampling.to_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv', index=False)
standard_sampling.to_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪standard_sampling.csv', index=False)
```

6 6.0 Clustering Method

6.1 6.1 Gaussian Mixture Model (GMM)

Reference: - https://builtin.com/articles/gaussian-mixture-model#:~:text=A%20Gaussian%20mixture%20model%

Gaussian Mixture Model (GMM) is a probabilistic model used in machine learning and statistics for clustering and density estimation. It assumes that the data is generated from a mixture of several Gaussian (normal) distributions, each representing a different cluster.

Key Concepts of GMM: 1. Gaussian Distribution: - A Gaussian distribution (also called a normal distribution) is defined by two parameters: the mean ( ) and the standard deviation ( ), which describe the center and the spread of the data, respectively.

2. Mixture of Gaussian

- GMM assumes that the data is a mixture of several Gaussian distributions. Each Gaussian represents a cluster, and the overall distribution of the data is a weighted sum of these individual Gaussians.

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

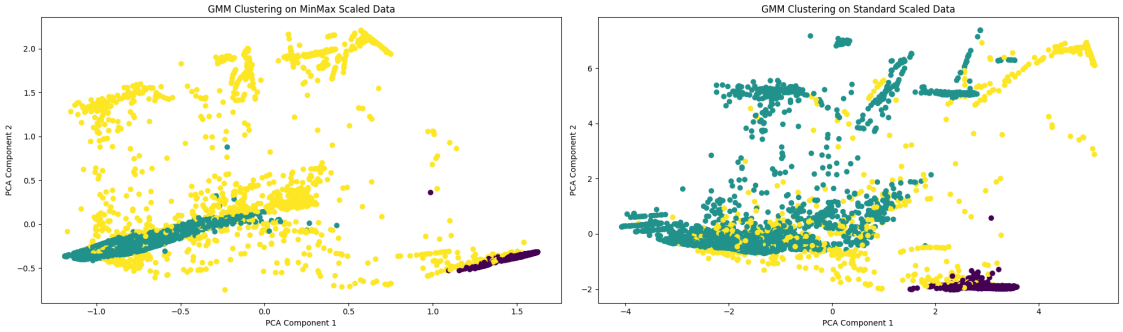
[ ]:
```
import pandas as pd

minmax_df = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv')
standard_df = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪standard_sampling.csv')
```

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score

selected_vars = [
    'count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'
]

minmax_data = minmax_df[selected_vars]
standard_data = standard_df[selected_vars]

# Number of clusters
n_clusters = 3

# GMM for MinMax Scaled Data
gmm_minmax = GaussianMixture(n_components=n_clusters, covariance_type='full',␣
 ↪random_state=42).fit(minmax_data)
minmax_labels = gmm_minmax.predict(minmax_data)

# GMM for Standard Scaled Data
```

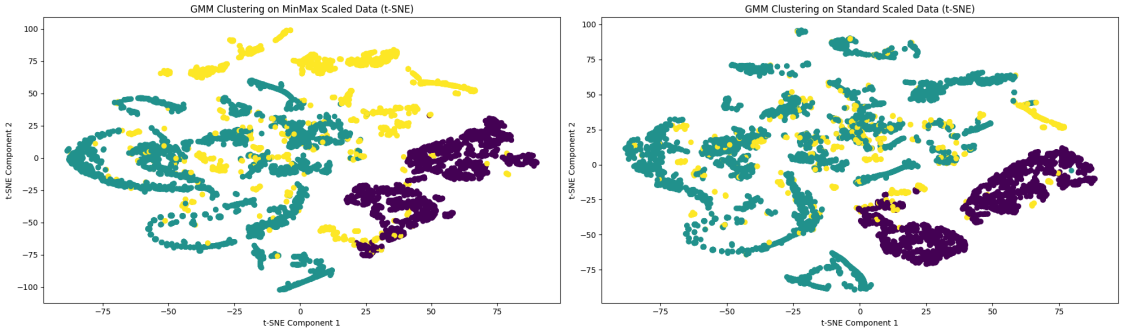
```
gmm_standard = GaussianMixture(n_components=n_clusters, covariance_type='full',␣
 ↪random_state=42).fit(standard_data)
standard_labels = gmm_standard.fit_predict(standard_data)

# Calculate Silhouette Scores
silhouette_minmax = silhouette_score(minmax_data, minmax_labels)
silhouette_standard = silhouette_score(standard_data, standard_labels)

# Display the Silhouette Scores
print(f'Silhouette Score for MinMax Scaled Data: {silhouette_minmax:.4f}')
print(f'Silhouette Score for Standard Scaled Data: {silhouette_standard:.4f}')
```

```
Silhouette Score for MinMax Scaled Data: 0.4704
Silhouette Score for Standard Scaled Data: 0.3441
```

[ ]:
```
# Use PCA for 2D Visualization
pca = PCA(n_components=2)
minmax_pca = pca.fit_transform(minmax_data)
standard_pca = pca.fit_transform(standard_data)

# Plotting the clusters for MinMax Scaled Data
plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 1)
plt.scatter(minmax_pca[:, 0], minmax_pca[:, 1], c=minmax_labels, cmap='viridis')
plt.title('GMM Clustering on MinMax Scaled Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

# Plotting the clusters for Standard Scaled Data
plt.subplot(1, 2, 2)
plt.scatter(standard_pca[:, 0], standard_pca[:, 1], c=standard_labels,␣
 ↪cmap='viridis')
plt.title('GMM Clustering on Standard Scaled Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

plt.tight_layout()
plt.show()
```



[ ]:
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Use t-SNE for 2D Visualization
tsne = TSNE(n_components=2, random_state=42)

# Apply t-SNE to both MinMax and Standard Scaled data
minmax_tsne = tsne.fit_transform(minmax_data)
standard_tsne = tsne.fit_transform(standard_data)

# Plotting the clusters for MinMax Scaled Data
plt.figure(figsize=(20, 6))

plt.subplot(1, 2, 1)
plt.scatter(minmax_tsne[:, 0], minmax_tsne[:, 1], c=minmax_labels,␣
 ↪cmap='viridis')
plt.title('GMM Clustering on MinMax Scaled Data (t-SNE)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')

# Plotting the clusters for Standard Scaled Data
plt.subplot(1, 2, 2)
plt.scatter(standard_tsne[:, 0], standard_tsne[:, 1], c=standard_labels,␣
 ↪cmap='viridis')
plt.title('GMM Clustering on Standard Scaled Data (t-SNE)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')

plt.tight_layout()
plt.show()
```



[ ]:
```
pip install umap
```

```
Requirement already satisfied: umap in /usr/local/lib/python3.10/dist-packages
(0.1.1)
```

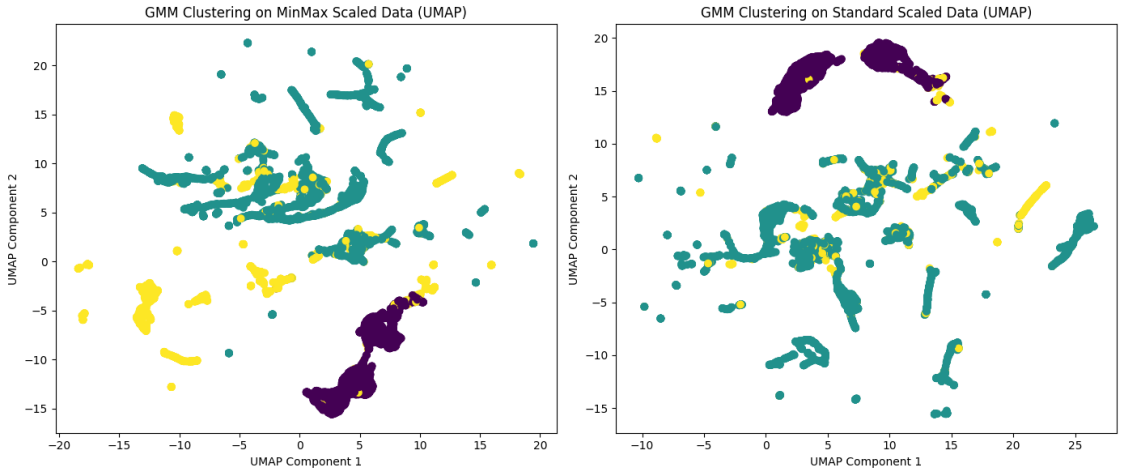
[ ]:
```
pip install umap-learn
```

```
Requirement already satisfied: umap-learn in /usr/local/lib/python3.10/dist-
packages (0.5.6)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (1.3.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (0.60.0)
Requirement already satisfied: pynndescent>=0.5 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (0.5.13)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from umap-learn) (4.66.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba>=0.51.2->umap-learn)
(0.43.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from pynndescent>=0.5->umap-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->umap-learn)
(3.5.0)
```

[ ]:
```
import umap.umap_ as umap
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
```

```
# Use UMAP for 2D Visualization
umap_reducer = umap.UMAP(n_components=2, random_state=42)
minmax_umap = umap_reducer.fit_transform(minmax_data)
standard_umap = umap_reducer.fit_transform(standard_data)

# Plotting the clusters for MinMax Scaled Data
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.scatter(minmax_umap[:, 0], minmax_umap[:, 1], c=minmax_labels,␣
 ↪cmap='viridis')
plt.title('GMM Clustering on MinMax Scaled Data (UMAP)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')

# Plotting the clusters for Standard Scaled Data
plt.subplot(1, 2, 2)
plt.scatter(standard_umap[:, 0], standard_umap[:, 1], c=standard_labels,␣
 ↪cmap='viridis')
plt.title('GMM Clustering on Standard Scaled Data (UMAP)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



6.1.1 6.1.1 Fine Tuning GMM

[ ]:
```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
import numpy as np

# Possible values for tuning
n_components_range = range(2, 6)
covariance_types = ['full', 'tied', 'diag', 'spherical']

# Function to perform GMM tuning and return the best model
def tune_gmm(data, data_name):
    best_gmm = None
    best_score = -1
    best_params = {}

    for n_components in n_components_range:
        for covariance_type in covariance_types:
            gmm = GaussianMixture(
                n_components=n_components,
                covariance_type=covariance_type,
                random_state=42
            )
            labels = gmm.fit_predict(data)
            score = silhouette_score(data, labels)

            # Save the best model
            if score > best_score:
                best_score = score
                best_gmm = gmm
                best_params = {
                    'n_components': n_components,
                    'covariance_type': covariance_type
                }

    print(f"Best Silhouette Score for {data_name}: {best_score:.4f}")
    print(f"Best Parameters for {data_name}: {best_params}")
    return best_gmm, best_score, best_params

# Tune GMM for MinMax scaled data
best_gmm_minmax, best_score_minmax, best_params_minmax = tune_gmm(minmax_data,␣
 ↪'MinMax Scaled Data')

# Tune GMM for Standard scaled data
best_gmm_standard, best_score_standard, best_params_standard =␣
 ↪tune_gmm(standard_data, 'Standard Scaled Data')
```

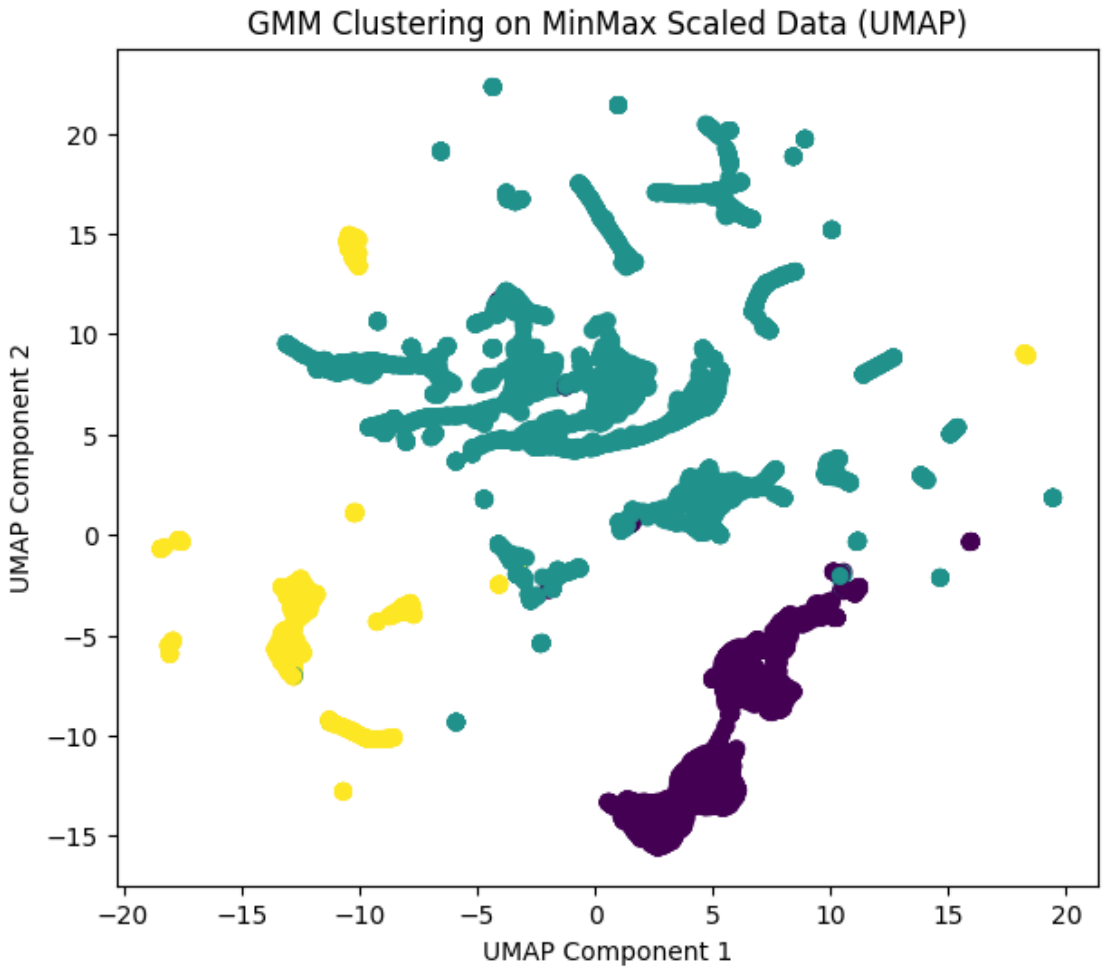
```
Best Silhouette Score for MinMax Scaled Data: 0.5916
```

```
Best Parameters for MinMax Scaled Data: {'n_components': 3, 'covariance_type':
'tied'}
Best Silhouette Score for Standard Scaled Data: 0.4644
Best Parameters for Standard Scaled Data: {'n_components': 3, 'covariance_type':
'tied'}
```

[ ]:
```python
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.manifold import TSNE
import umap
import matplotlib.pyplot as plt

# Function to apply dimensionality reduction and visualize clustering
def visualize_clustering_and_silhouette(data, labels, title, method='PCA'):
    if method == 'PCA':
        reducer = PCA(n_components=2, random_state=42)
    elif method == 'UMAP':
        reducer = umap.UMAP(n_components=2, random_state=42)
    elif method == 't-SNE':
        reducer = TSNE(n_components=2, random_state=42)

    # Apply dimensionality reduction
    reduced_data = reducer.fit_transform(data)

    # Plotting the clusters
    plt.figure(figsize=(7, 6))
    plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=labels,␣
 ↪cmap='viridis')
    plt.title(f'GMM Clustering on {title} ({method})')
    plt.xlabel(f'{method} Component 1')
    plt.ylabel(f'{method} Component 2')
    plt.show()

    # Calculate and print silhouette score
    silhouette_avg = silhouette_score(data, labels)
    print(f'Silhouette Score for {title} ({method}): {silhouette_avg:.4f}')
```

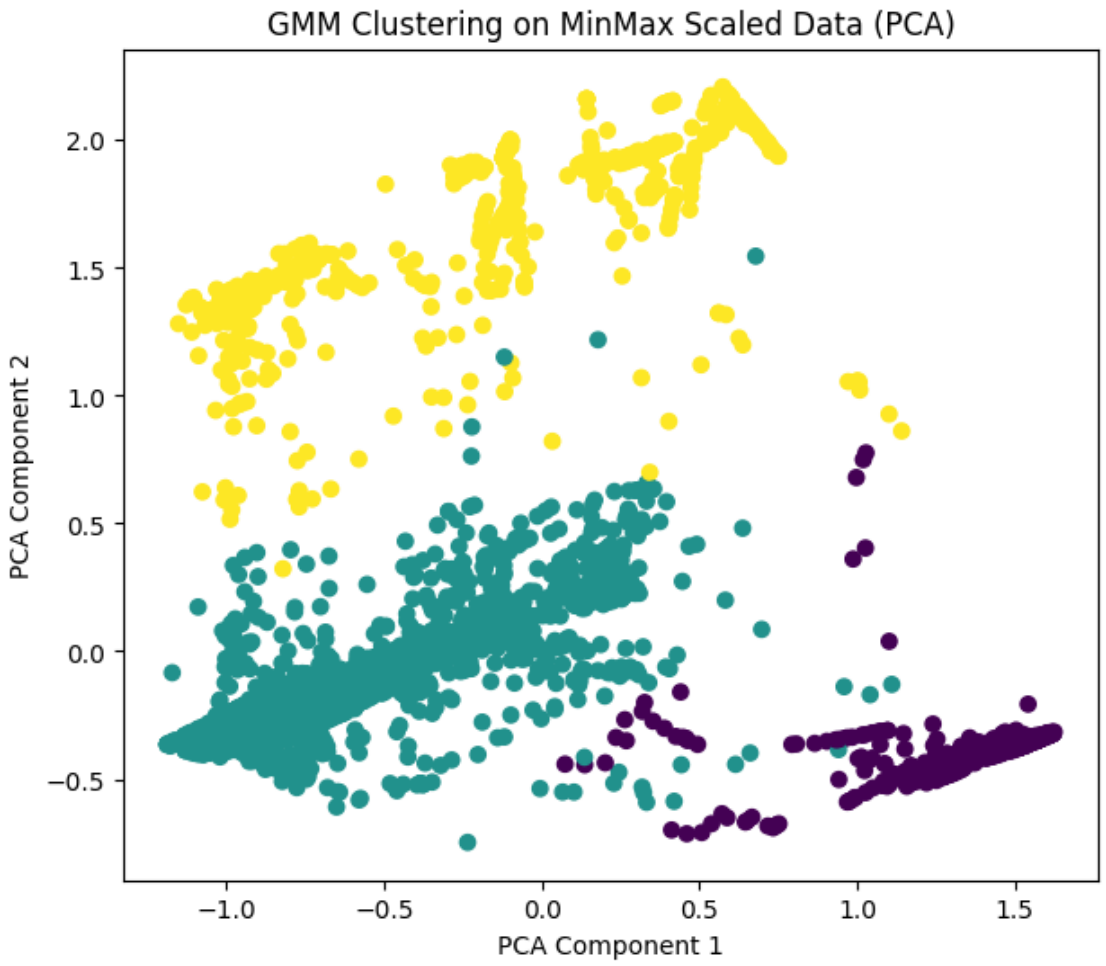
[ ]:
```python
# Apply GMM to MinMax scaled data and visualize with UMAP and PCA
labels_minmax = best_gmm_minmax.predict(minmax_data)
visualize_clustering_and_silhouette(minmax_data, labels_minmax, 'MinMax Scaled␣
 ↪Data', method='UMAP')
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



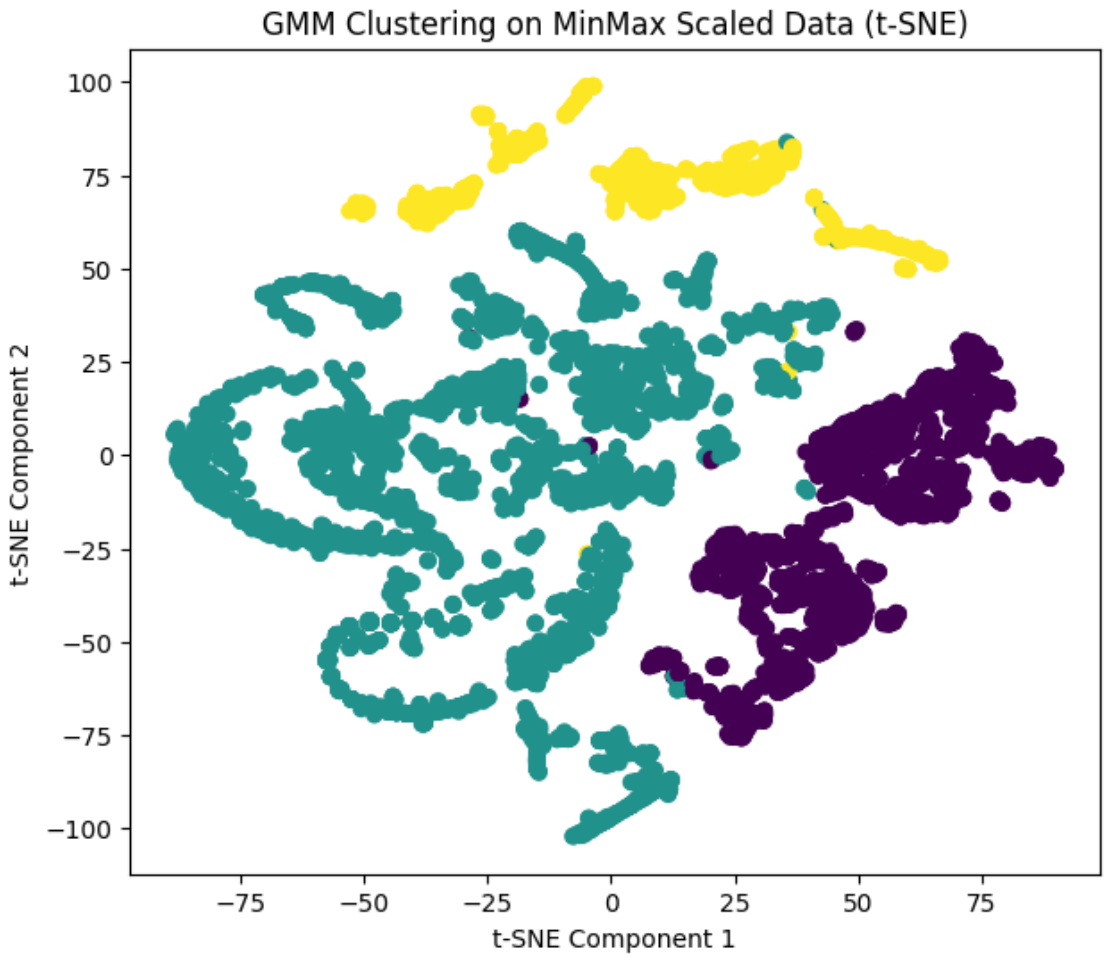

```
Silhouette Score for MinMax Scaled Data (UMAP): 0.5916
```

[ ]:
```
visualize_clustering_and_silhouette(minmax_data, labels_minmax, 'MinMax Scaled␣
↪Data', method='PCA')
```




```
Silhouette Score for MinMax Scaled Data (PCA): 0.5916
```

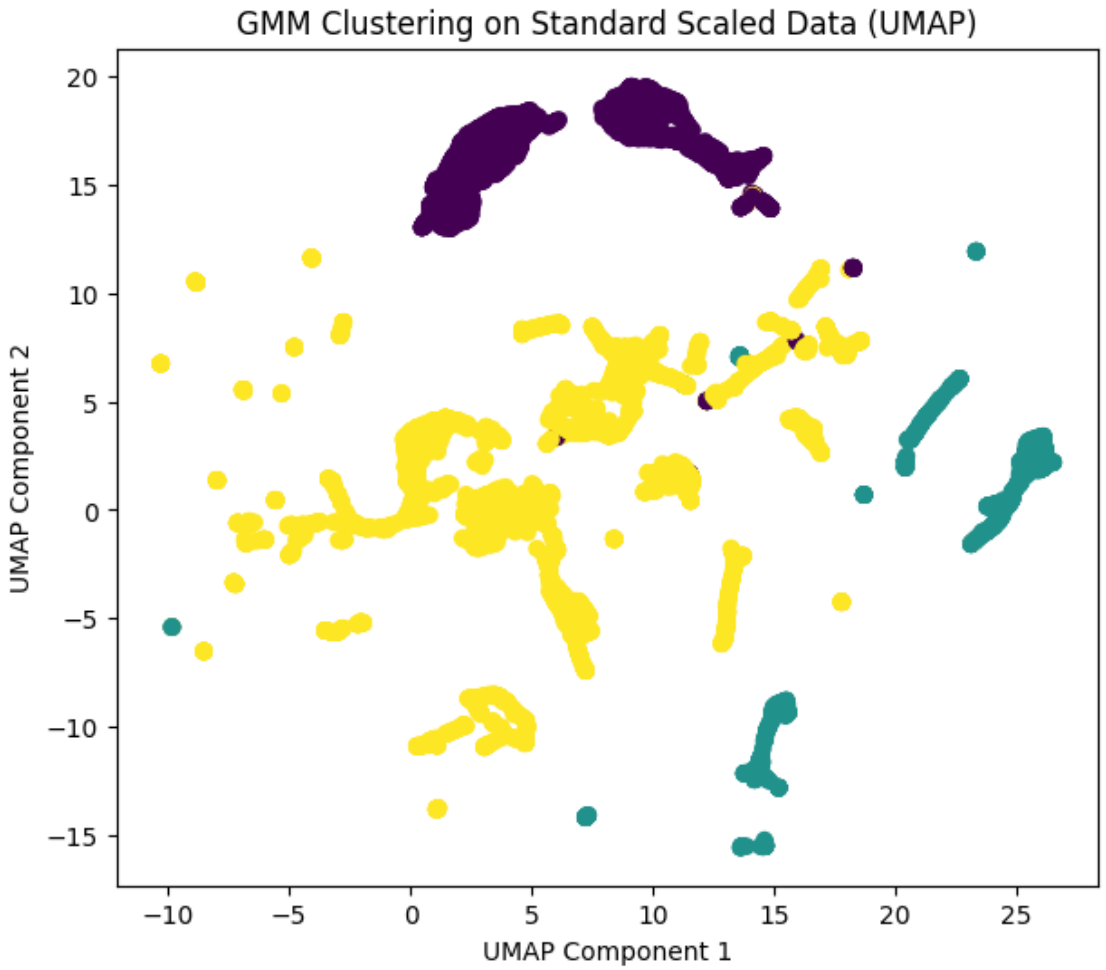
[ ]:
```
visualize_clustering_and_silhouette(minmax_data, labels_minmax, 'MinMax Scaled␣
↪Data', method='t-SNE')
```




```
Silhouette Score for MinMax Scaled Data (t-SNE): 0.5916
```

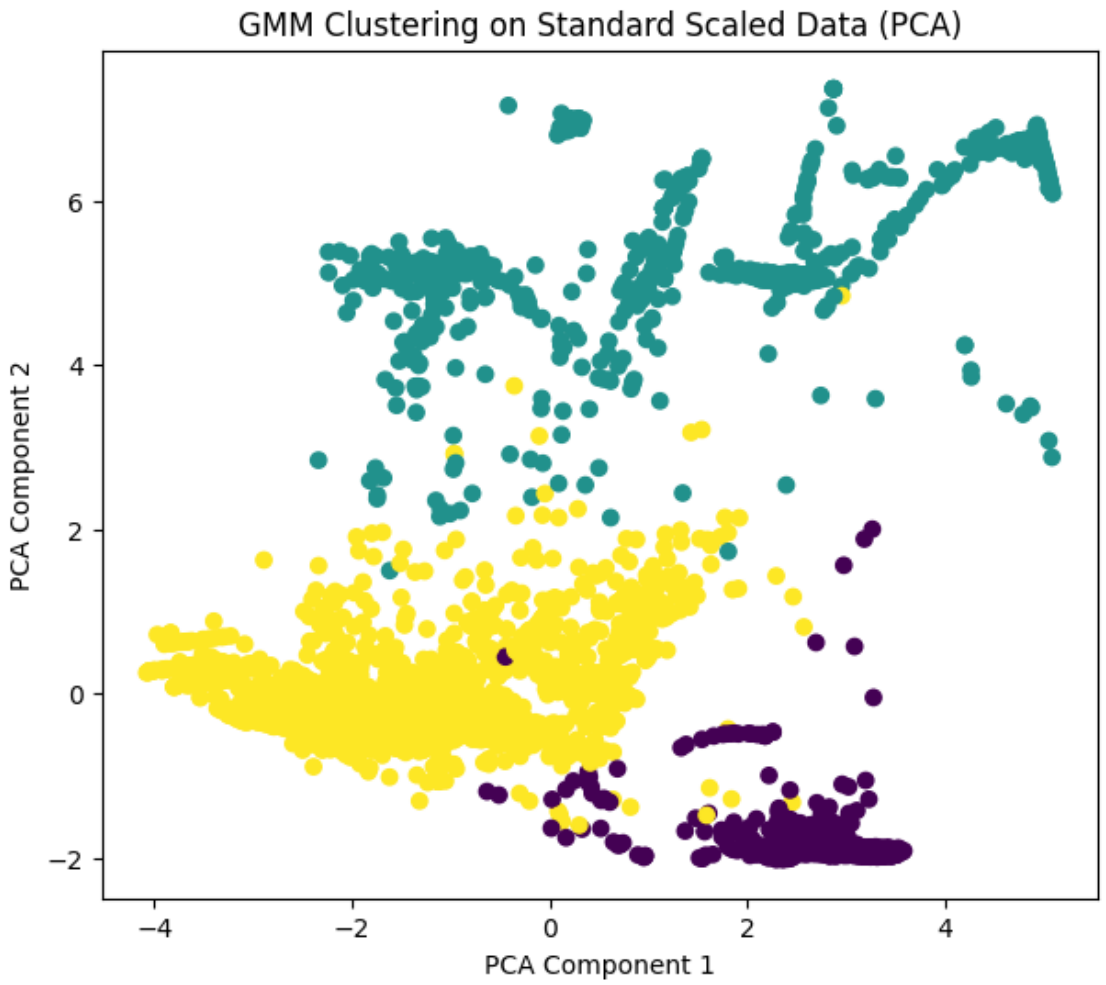
[ ]:
```
# Apply GMM to Standard scaled data and visualize with UMAP and PCA
labels_standard = best_gmm_standard.predict(standard_data)
visualize_clustering_and_silhouette(standard_data, labels_standard, 'Standard␣
↪Scaled Data', method='UMAP')
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



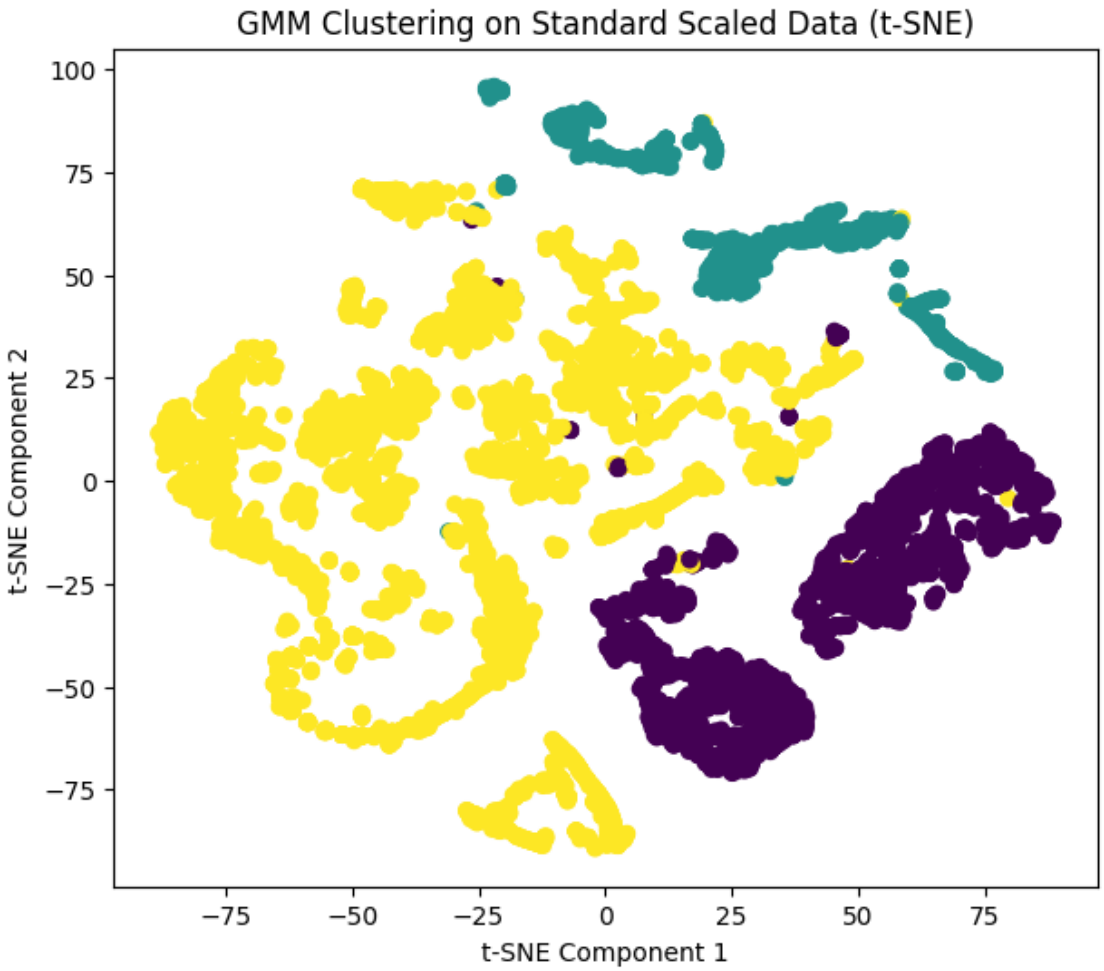

```
Silhouette Score for Standard Scaled Data (UMAP): 0.4644
```

```
[ ]: visualize_clustering_and_silhouette(standard_data, labels_standard, 'Standard␣
  ↪Scaled Data', method='PCA')
```




```
Silhouette Score for Standard Scaled Data (PCA): 0.4644
```

[ ]:
```
visualize_clustering_and_silhouette(standard_data, labels_standard, 'Standard␣
↪Scaled Data', method='t-SNE')
```




```
Silhouette Score for Standard Scaled Data (t-SNE): 0.4644
```

6.1.2 6.1.2 Evaluation after tuning

- Calinski-Harabasz Index (Variance Ratio Criterion)

[ ]:
```
from sklearn.metrics import calinski_harabasz_score

ch_score = calinski_harabasz_score(minmax_data, labels_minmax)
print(f'Calinski-Harabasz Score (MinMax): {ch_score:.4f}')

ch_score_1 = calinski_harabasz_score(standard_data, labels_standard)
print(f'Calinski-Harabasz Score (Standard): {ch_score_1:.4f}')
```

```
Calinski-Harabasz Score (MinMax): 8404.6057
Calinski-Harabasz Score (Standard): 2816.6795
```

- Davies-Bouldin Index

[ ]:
```
from sklearn.metrics import davies_bouldin_score

db_score = davies_bouldin_score(minmax_data, labels_minmax)
print(f'Davies-Bouldin Score (MinMax): {db_score:.4f}')

db_score_1 = davies_bouldin_score(standard_data, labels_standard)
print(f'Davies-Bouldin Score (Standard): {db_score_1:.4f}')
```

```
Davies-Bouldin Score (MinMax): 0.7371
Davies-Bouldin Score (Standard): 0.9173
```

- BIC/AIC Scores

[ ]:
```
bic = best_gmm_minmax.bic(minmax_data)
aic = best_gmm_minmax.aic(minmax_data)
print("Min Max Scaling")
print(f'BIC: {bic:.4f}, AIC: {aic:.4f}')


bic_1 = best_gmm_minmax.bic(standard_data)
aic_1 = best_gmm_minmax.aic(standard_data)
print('Standard Scaling')
print(f'BIC: {bic_1:.4f}, AIC: {aic_1:.4f}')
```

```
Min Max Scaling
BIC: -709478.9035, AIC: -712307.3224
Standard Scaling
BIC: 1273559178.8685, AIC: 1273556350.4496
```

6.1.3 6.1.3 Summary of Comparison

1. Calinski-Harabasz Score

- MinMax Scaling: 8404.6057
- Standard Scaling: 2816.6795
- Interpretation: A higher Calinski-Harabasz score indicates better-defined clusters. MinMax Scaling significantly outperforms Standard Scaling in this metric.

2. Davies-Bouldin Score

- MinMax Scaling: 0.7371
- Standard Scaling: 0.9173
- Interpretation: A lower Davies-Bouldin score is better, as it indicates lower within-cluster variance relative to the separation between clusters. MinMax Scaling again outperforms Standard Scaling here.

3. Bayesian Information Criterion (BIC) & Akaike Information Criterion (AIC)

- MinMax Scaling BIC: -709478.9035
- Standard Scaling BIC: 1273559178.8685
- MinMax Scaling AIC: -712307.3224

- Standard Scaling AIC: 1273556350.4496
- Interpretation: Lower BIC and AIC values indicate a better model fit with fewer parameters. MinMax Scaling has much lower (negative) BIC and AIC values compared to Standard Scaling, indicating a better model fit.

4. Silhouette Score

- MinMax Scaling (UMAP & PCA): 0.5916
- Standard Scaling (UMAP & PCA): 0.4644
- Interpretation: The Silhouette Score measures how similar an object is to its own cluster compared to other clusters. A higher score indicates better clustering. MinMax Scaling provides a significantly higher score, indicating more coherent clusters.

Conclusion: MinMax Scaling is the better scaling method based on all the evaluation metrics. It provides better-defined clusters (higher Calinski-Harabasz score), lower within-cluster variance relative to separation (lower Davies-Bouldin score), a better model fit (lower BIC and AIC), and more coherent clusters (higher Silhouette Score).

6.2 6.2 HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that extends the popular DBSCAN algorithm by incorporating hierarchical clustering concepts. It is primarily used for unsupervised learning tasks where the goal is to group data points into clusters based on their density, with the added benefit of identifying clusters of varying densities and handling noise.

Key Features of HDBSCAN: 1. Density-Based Clustering: - HDBSCAN groups points that are close together in high-density regions. Unlike K-Means, where the number of clusters must be predefined, HDBSCAN automatically finds the optimal number of clusters based on the data's density.

2. Variable Density Clusters:

- One of the main advantages of HDBSCAN over DBSCAN is its ability to find clusters with varying densities. This allows HDBSCAN to handle more complex clustering tasks, where clusters have different levels of density, which DBSCAN cannot do well.

3. Hierarchical Clustering:

- HDBSCAN builds a hierarchy of clusters using a technique called "single linkage clustering," from which the best clustering can be selected based on stability. The hierarchy provides a multi-level view of the clustering structure.

4. Noise Handling:

- HDBSCAN can automatically classify points as noise (i.e., points that do not belong to any cluster). This makes it effective at identifying outliers and irrelevant points in datasets.

5. No Need to Predefine the Number of Clusters:

- Unlike K-Means and other clustering algorithms, HDBSCAN does not require you to specify the number of clusters beforehand. Instead, it finds the appropriate number of clusters based on data properties.

[ ]:
```
import pandas as pd

minmax_df = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv')
standard_df = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪standard_sampling.csv')
```

[ ]:
```
pip install hdbscan
```

```
Requirement already satisfied: hdbscan in /usr/local/lib/python3.10/dist-
packages (0.8.38.post1)
Requirement already satisfied: numpy<3,>=1.20 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.26.4)
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.13.1)
Requirement already satisfied: scikit-learn>=0.20 in
/usr/local/lib/python3.10/dist-packages (from hdbscan) (1.3.2)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.10/dist-
packages (from hdbscan) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20->hdbscan)
(3.5.0)
```

[ ]:
```
import hdbscan
import pandas as pd
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import silhouette_score

selected_vars = [
    'count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'
]

minmax_data = minmax_df[selected_vars]
standard_data = standard_df[selected_vars]

# HDBSCAN model
hdbscan_model = hdbscan.HDBSCAN(min_cluster_size=3, min_samples=1000)

# Fit HDBSCAN on MinMax Scaled Data
hdbscan_minmax_labels = hdbscan_model.fit_predict(minmax_data)
```

```
# Fit HDBSCAN on Standard Scaled Data
hdbscan_standard_labels = hdbscan_model.fit_predict(standard_data)

# Calculate Silhouette Scores
silhouette_minmax = silhouette_score(minmax_data, hdbscan_minmax_labels)
silhouette_standard = silhouette_score(standard_data, hdbscan_standard_labels)

# Display the Silhouette Scores
print(f'Silhouette Score for MinMax Scaled Data (HDBSCAN): {silhouette_minmax:.
  ↪4f}')
print(f'Silhouette Score for Standard Scaled Data (HDBSCAN):␣
  ↪{silhouette_standard:.4f}')
```

```
Silhouette Score for MinMax Scaled Data (HDBSCAN): 0.4643
Silhouette Score for Standard Scaled Data (HDBSCAN): 0.3504
```

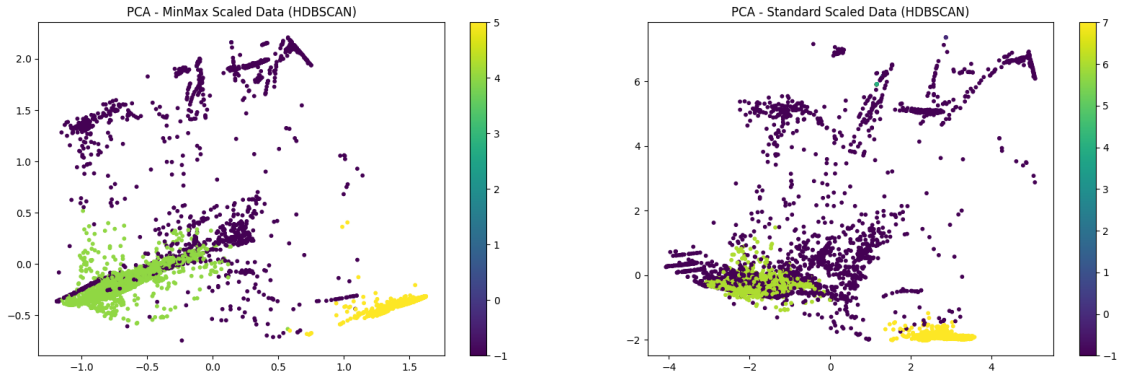
[ ]:
```
pip install umap
```

```
Requirement already satisfied: umap in /usr/local/lib/python3.10/dist-packages
(0.1.1)
```

[ ]:
```
pip install umap-learn
```

```
Requirement already satisfied: umap-learn in /usr/local/lib/python3.10/dist-
packages (0.5.6)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (1.3.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (0.60.0)
Requirement already satisfied: pynndescent>=0.5 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (0.5.13)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from umap-learn) (4.66.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba>=0.51.2->umap-learn)
(0.43.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from pynndescent>=0.5->umap-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->umap-learn)
(3.5.0)
```

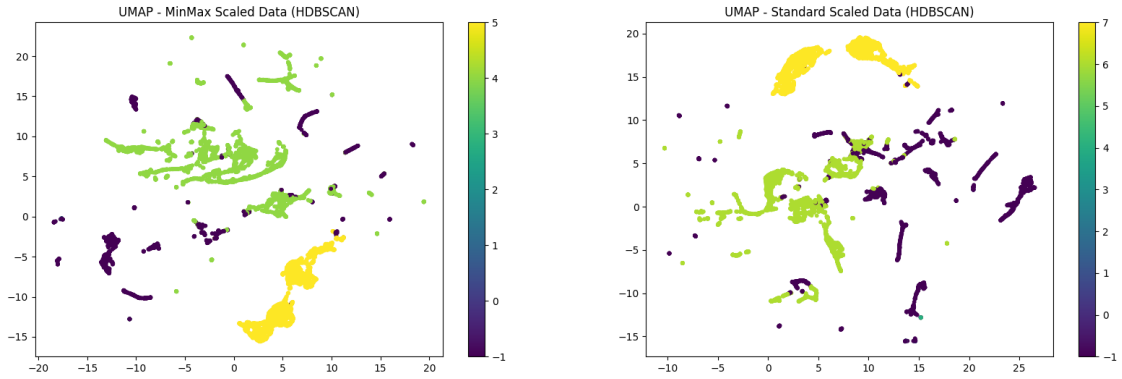
[ ]:
```
from sklearn.decomposition import PCA
import umap
import matplotlib.pyplot as plt

# Apply PCA to reduce to 2 components for visualization
pca_minmax = PCA(n_components=2).fit_transform(minmax_data)
pca_standard = PCA(n_components=2).fit_transform(standard_data)

# Plot PCA results for MinMax Scaled Data
plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 1)
plt.scatter(pca_minmax[:, 0], pca_minmax[:, 1], c=hdbscan_minmax_labels,␣
 ↪cmap='viridis', s=10)
plt.title('PCA - MinMax Scaled Data (HDBSCAN)')
plt.colorbar()

# Plot PCA results for Standard Scaled Data
plt.subplot(1, 2, 2)
plt.scatter(pca_standard[:, 0], pca_standard[:, 1], c=hdbscan_standard_labels,␣
 ↪cmap='viridis', s=10)
plt.title('PCA - Standard Scaled Data (HDBSCAN)')
plt.colorbar()
plt.show()
```



[ ]:
```
# Apply UMAP to reduce to 2 components for visualization
umap_model = umap.UMAP(n_components=2, random_state=42)

umap_minmax = umap_model.fit_transform(minmax_data)
umap_standard = umap_model.fit_transform(standard_data)

# Plot UMAP results for MinMax Scaled Data
plt.figure(figsize=(20, 6))
plt.subplot(1, 2, 1)
```

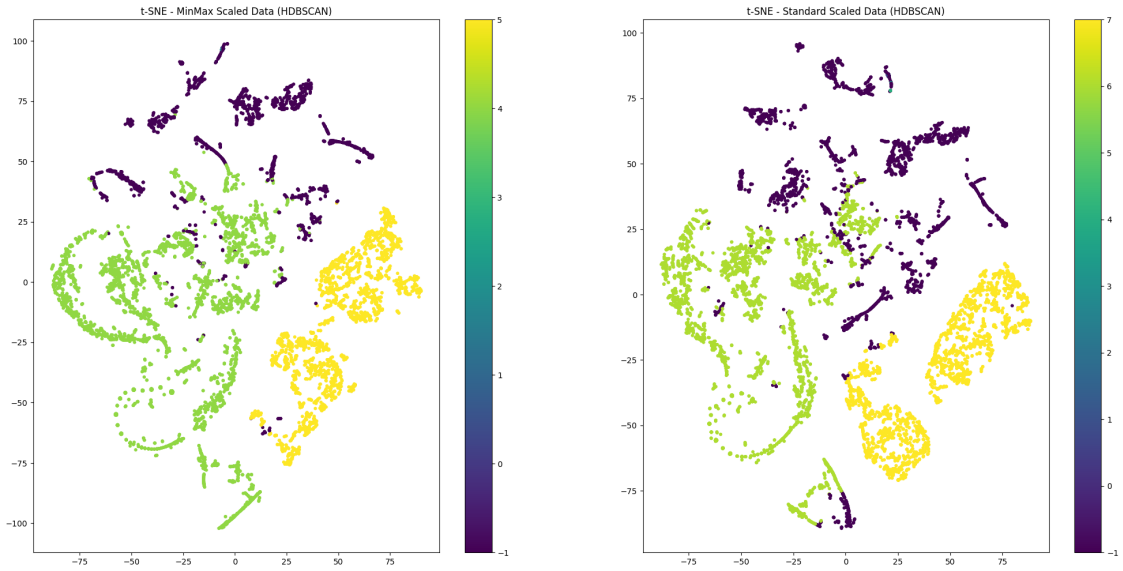
```
plt.scatter(umap_minmax[:, 0], umap_minmax[:, 1], c=hdbscan_minmax_labels,␣
 ↪cmap='viridis', s=10)
plt.title('UMAP - MinMax Scaled Data (HDBSCAN)')
plt.colorbar()

# Plot UMAP results for Standard Scaled Data
plt.subplot(1, 2, 2)
plt.scatter(umap_standard[:, 0], umap_standard[:, 1],␣
 ↪c=hdbscan_standard_labels, cmap='viridis', s=10)
plt.title('UMAP - Standard Scaled Data (HDBSCAN)')
plt.colorbar()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



[ ]:
```
from sklearn.manifold import TSNE

# Apply t-SNE to reduce to 2 components for visualization
tsne_minmax = TSNE(n_components=2, random_state=42).fit_transform(minmax_data)
tsne_standard = TSNE(n_components=2, random_state=42).
 ↪fit_transform(standard_data)

# Plot t-SNE results for MinMax Scaled Data
plt.figure(figsize=(25, 12))
plt.subplot(1, 2, 1)
plt.scatter(tsne_minmax[:, 0], tsne_minmax[:, 1], c=hdbscan_minmax_labels,␣
 ↪cmap='viridis', s=10)
plt.title('t-SNE - MinMax Scaled Data (HDBSCAN)')
plt.colorbar()
```

```
# Plot t-SNE results for Standard Scaled Data
plt.subplot(1, 2, 2)
plt.scatter(tsne_standard[:, 0], tsne_standard[:, 1],␣
 ↪c=hdbscan_standard_labels, cmap='viridis', s=10)
plt.title('t-SNE - Standard Scaled Data (HDBSCAN)')
plt.colorbar()
plt.show()
```



6.2.1 6.2.1 fine tuning HDBSCAN

[ ]:
```
import hdbscan
from sklearn.metrics import silhouette_score
import numpy as np

# Function to evaluate HDBSCAN with different hyperparameters
def tune_hdbscan(data, min_cluster_size_range, min_samples_range,␣
 ↪epsilon_range):
    best_silhouette = -1
    best_params = {}

    for min_cluster_size in min_cluster_size_range:
        for min_samples in min_samples_range:
            for epsilon in epsilon_range:
                # Initialize HDBSCAN with current parameters
                hdbscan_model = hdbscan.
 ↪HDBSCAN(min_cluster_size=min_cluster_size,
                                                    min_samples=min_samples,
```

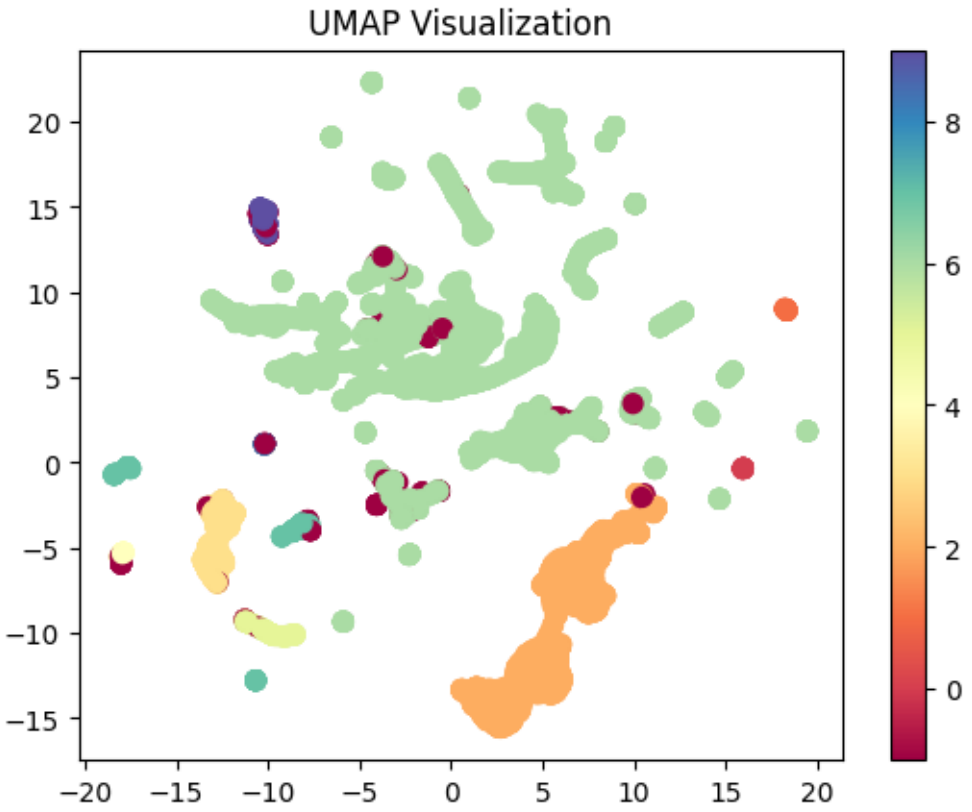
```
                                                    ␣
 ↪cluster_selection_epsilon=epsilon)

                # Fit and predict
                labels = hdbscan_model.fit_predict(data)

                # Calculate silhouette score (only if more than one cluster is␣
 ↪found)
                if len(np.unique(labels)) > 1:
                    silhouette_avg = silhouette_score(data, labels)

                    # Update best parameters if current score is better
                    if silhouette_avg > best_silhouette:
                        best_silhouette = silhouette_avg
                        best_params = {
                            'min_cluster_size': min_cluster_size,
                            'min_samples': min_samples,
                            'cluster_selection_epsilon': epsilon
                        }

    return best_silhouette, best_params

# Define ranges for tuning
min_cluster_size_range = [3, 5, 10]
min_samples_range = [5, 10, 15]
epsilon_range = [0.1, 0.5, 1.0]

# Perform tuning for both datasets
silhouette_minmax, best_params_minmax = tune_hdbscan(minmax_data,␣
 ↪min_cluster_size_range, min_samples_range, epsilon_range)
silhouette_standard, best_params_standard = tune_hdbscan(standard_data,␣
 ↪min_cluster_size_range, min_samples_range, epsilon_range)

# Display the results
print(f'Best Silhouette Score for MinMax Scaled Data: {silhouette_minmax:.4f}')
print(f'Best Parameters for MinMax Scaled Data: {best_params_minmax}')

print(f'Best Silhouette Score for Standard Scaled Data: {silhouette_standard:.
 ↪4f}')
print(f'Best Parameters for Standard Scaled Data: {best_params_standard}')
```

```
Best Silhouette Score for MinMax Scaled Data: 0.5446
Best Parameters for MinMax Scaled Data: {'min_cluster_size': 10, 'min_samples':
10, 'cluster_selection_epsilon': 0.5}
Best Silhouette Score for Standard Scaled Data: 0.3561
Best Parameters for Standard Scaled Data: {'min_cluster_size': 5, 'min_samples':
15, 'cluster_selection_epsilon': 1.0}
```

[ ]:
```
import umap
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# Function to visualize clustering using PCA, UMAP, and t-SNE
def visualize_clusters(data, labels, method="PCA"):
    if method == "PCA":
        # PCA for 2D visualization
        pca = PCA(n_components=2)
        components = pca.fit_transform(data)
        plt.title("PCA Visualization")
    elif method == "UMAP":
        # UMAP for 2D visualization
        reducer = umap.UMAP(n_components=2, random_state=42)
        components = reducer.fit_transform(data)
        plt.title("UMAP Visualization")
    elif method == "t-SNE":
        # t-SNE for 2D visualization
        tsne = TSNE(n_components=2, random_state=42)
        components = tsne.fit_transform(data)
        plt.title("t-SNE Visualization")

    # Plotting the visualization
    plt.scatter(components[:, 0], components[:, 1], c=labels, cmap='Spectral',␣
 ↪s=50)
    plt.colorbar()
    plt.show()

# After tuning HDBSCAN on MinMax Scaled Data
hdbscan_model_minmax = hdbscan.
 ↪HDBSCAN(min_cluster_size=best_params_minmax['min_cluster_size'],
                                        ␣
 ↪min_samples=best_params_minmax['min_samples'],
                                        ␣
 ↪cluster_selection_epsilon=best_params_minmax['cluster_selection_epsilon'])
hdbscan_minmax_labels = hdbscan_model_minmax.fit_predict(minmax_data)

# After tuning HDBSCAN on Standard Scaled Data
hdbscan_model_standard = hdbscan.
 ↪HDBSCAN(min_cluster_size=best_params_standard['min_cluster_size'],
                                         ␣
 ↪min_samples=best_params_standard['min_samples'],
                                         ␣
 ↪cluster_selection_epsilon=best_params_standard['cluster_selection_epsilon'])
hdbscan_standard_labels = hdbscan_model_standard.fit_predict(standard_data)
```

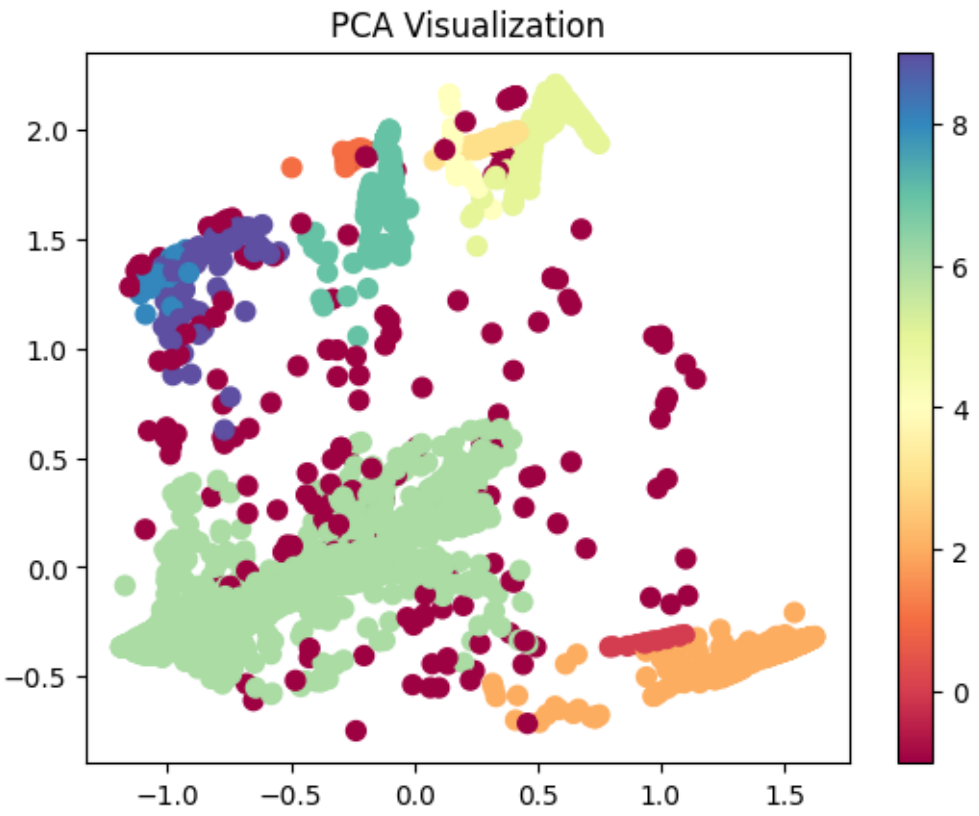
```
# Visualize the clustering for MinMax Scaled Data
print("UMAP Visualization for MinMax Scaled Data:")
visualize_clusters(minmax_data, hdbscan_minmax_labels, method="UMAP")
```

```
UMAP Visualization for MinMax Scaled Data:

/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



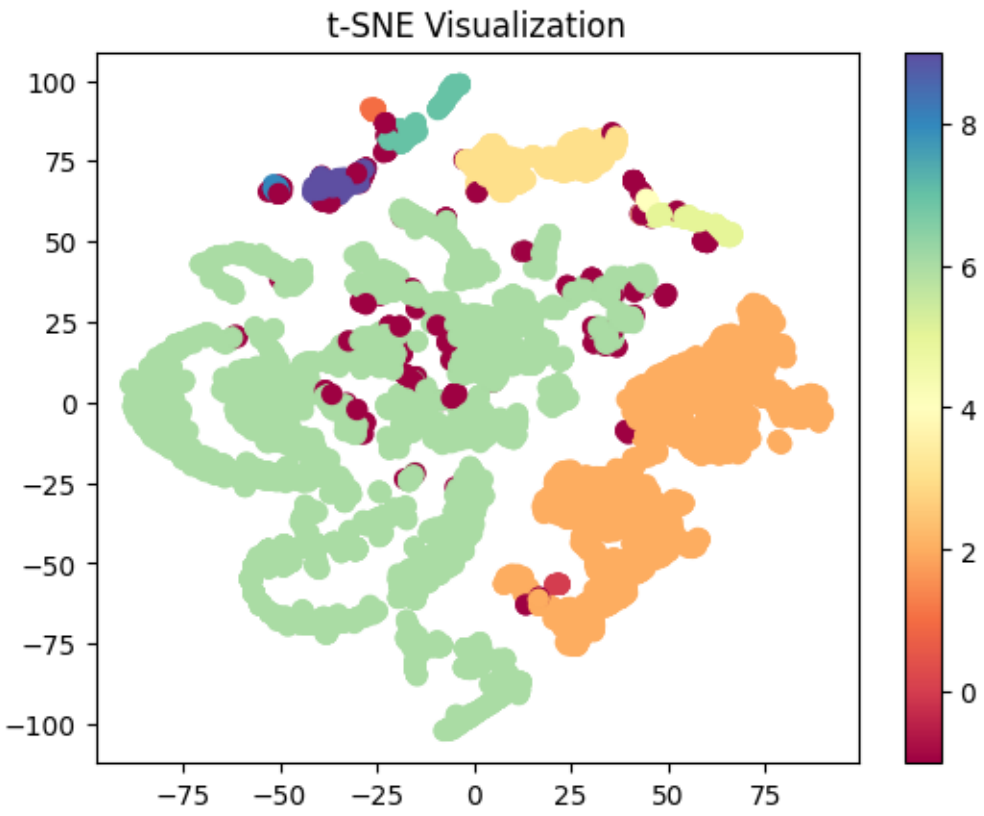
[ ]:
```
print("PCA Visualization for MinMax Scaled Data:")
visualize_clusters(minmax_data, hdbscan_minmax_labels, method="PCA")
```

```
PCA Visualization for MinMax Scaled Data:
```



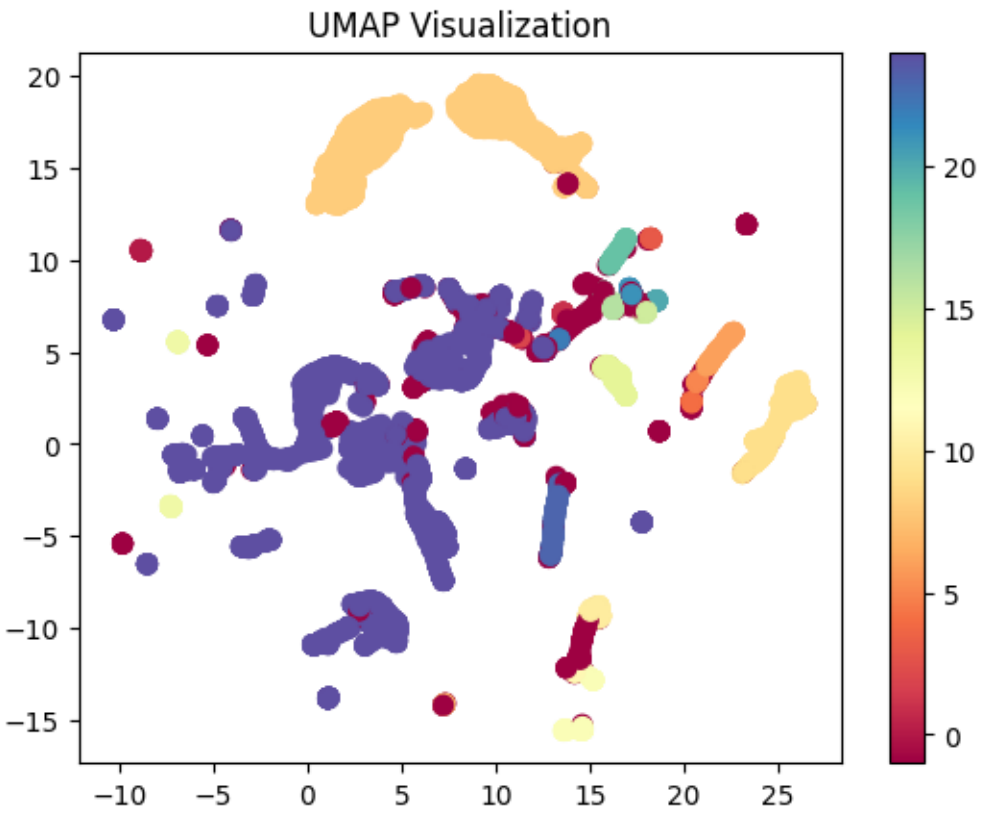
[ ]:
```
print("t-SNE Visualization for MinMax Scaled Data:")
visualize_clusters(minmax_data, hdbscan_minmax_labels, method="t-SNE")
```

```
t-SNE Visualization for MinMax Scaled Data:
```



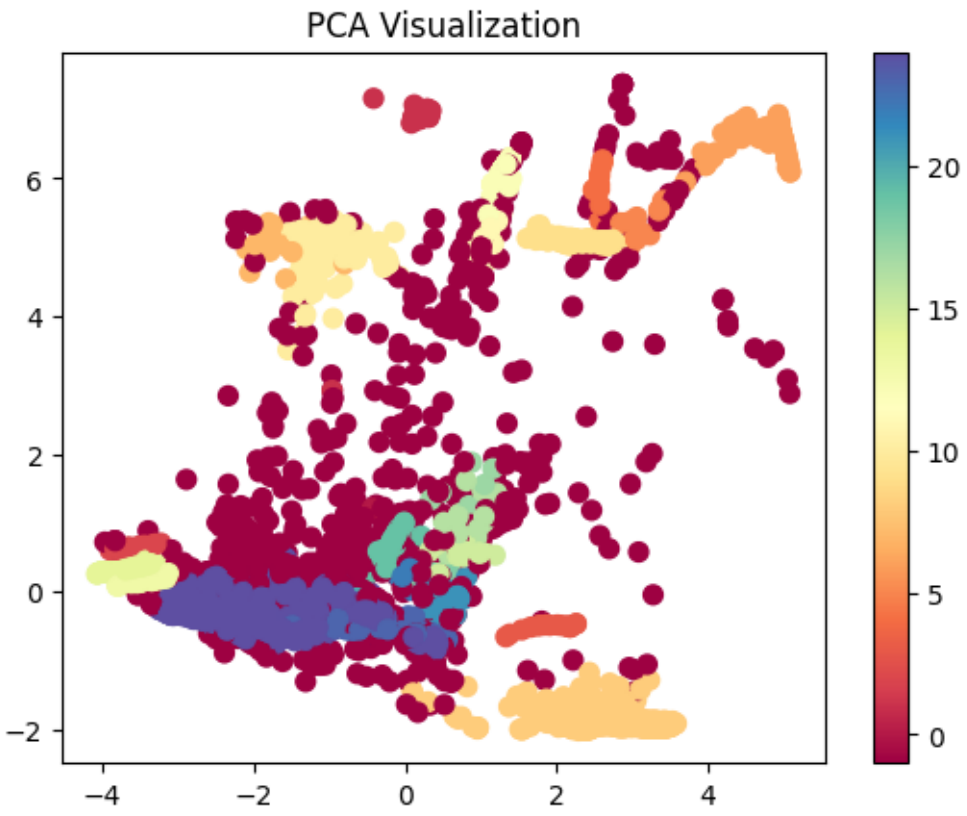
[ ]:
```
# Visualize the clustering for Standard Scaled Data
print("UMAP Visualization for Standard Scaled Data:")
visualize_clusters(standard_data, hdbscan_standard_labels, method="UMAP")
```

```
UMAP Visualization for Standard Scaled Data:

/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



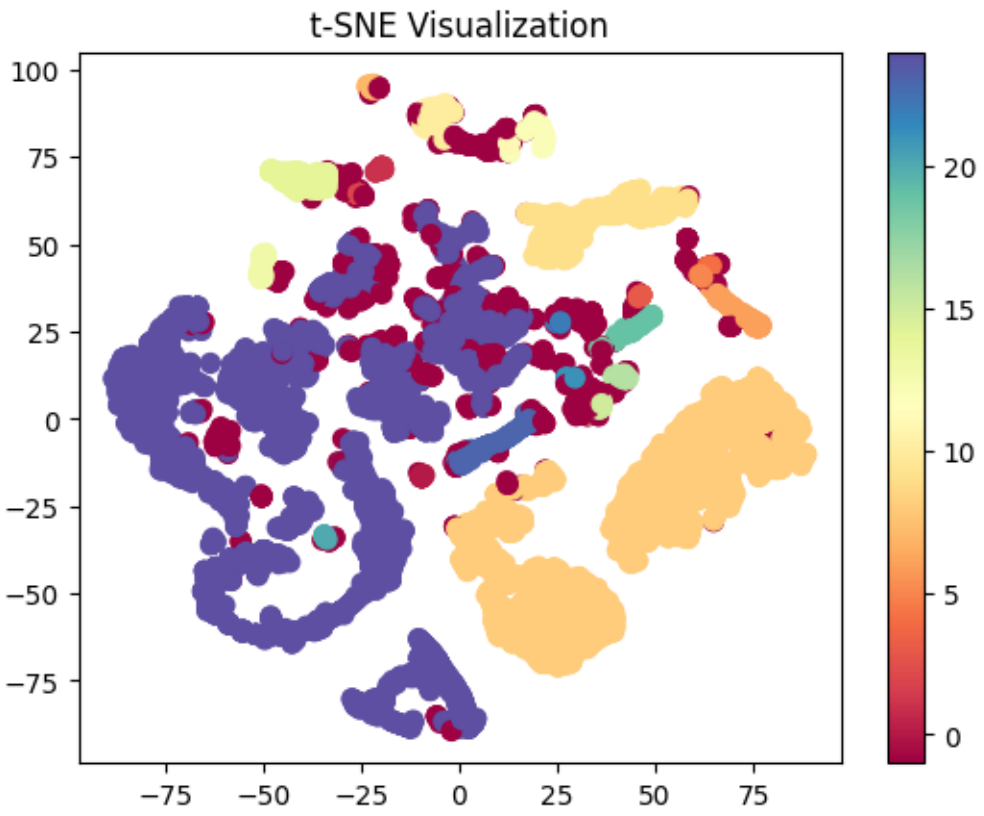
[ ]:
```
print("PCA Visualization for Standard Scaled Data:")
visualize_clusters(standard_data, hdbscan_standard_labels, method="PCA")
```

```
PCA Visualization for Standard Scaled Data:
```



[ ]:
```
print("t-SNE Visualization for Standard Scaled Data:")
visualize_clusters(standard_data, hdbscan_standard_labels, method="t-SNE")
```

```
t-SNE Visualization for Standard Scaled Data:
```



6.2.2 6.2.2 Evaluation after tuning

[ ]:
```
from sklearn.metrics import calinski_harabasz_score

ch_score_minmax = calinski_harabasz_score(minmax_data, hdbscan_minmax_labels)
ch_score_standard = calinski_harabasz_score(standard_data,␣
 ↪hdbscan_standard_labels)

print(f"MinMax Scaled Data - Calinski-Harabasz Score: {ch_score_minmax:.4f}")
print(f"Standard Scaled Data - Calinski-Harabasz Score: {ch_score_standard:.
 ↪4f}")
```

```
MinMax Scaled Data - Calinski-Harabasz Score: 2173.0042
Standard Scaled Data - Calinski-Harabasz Score: 465.6494
```

[ ]:
```
from sklearn.metrics import davies_bouldin_score

 db_score_minmax = davies_bouldin_score(minmax_data, hdbscan_minmax_labels)
 db_score_standard = davies_bouldin_score(standard_data,␣
 ↪hdbscan_standard_labels)
```

```
print(f"MinMax Scaled Data - Davies-Bouldin Score: {db_score_minmax:.4f}")
print(f"Standard Scaled Data - Davies-Bouldin Score: {db_score_standard:.4f}")
```

```
MinMax Scaled Data - Davies-Bouldin Score: 1.2836
Standard Scaled Data - Davies-Bouldin Score: 1.2979
```

6.2.3 6.2.3 Summary of Comparison

1. Silhouette Score

- MinMax Scaled Data: 0.5446
- Standard Scaled Data: 0.3561
- Interpretation: The Silhouette Score measures how similar a point is to its own cluster compared to other clusters. The higher the score, the better the clusters are defined.

2. Calinski-Harabasz Score

- MinMax Scaled Data: 2173.0042
- Standard Scaled Data: 465.6494
- Interpretation: The Calinski-Harabasz Index evaluates cluster dispersion; a higher value indicates that the clusters are dense and well-separated.

3. Davies-Bouldin Score

- MinMax Scaled Data: 1.2836
- Standard Scaled Data: 1.2979
- Interpretation: The Davies-Bouldin Index evaluates the ratio of within-cluster scatter to between-cluster separation; a lower score is better.

Conclusion:

- MinMax Scaled Data is overall the better dataset for clustering, based on the Silhouette Score and Calinski-Harabasz Score. These metrics emphasize that the clusters in this dataset are more compact and better defined.
- While Standard Scaled Data has a marginally better Davies-Bouldin Score, this does not outweigh the significantly better performance of MinMax Scaled Data in the other two key metrics.

6.3 6.3 DBSCAN

References :

- https://scikit-learn.org/dev/auto_examples/cluster/plot_dbscan.html

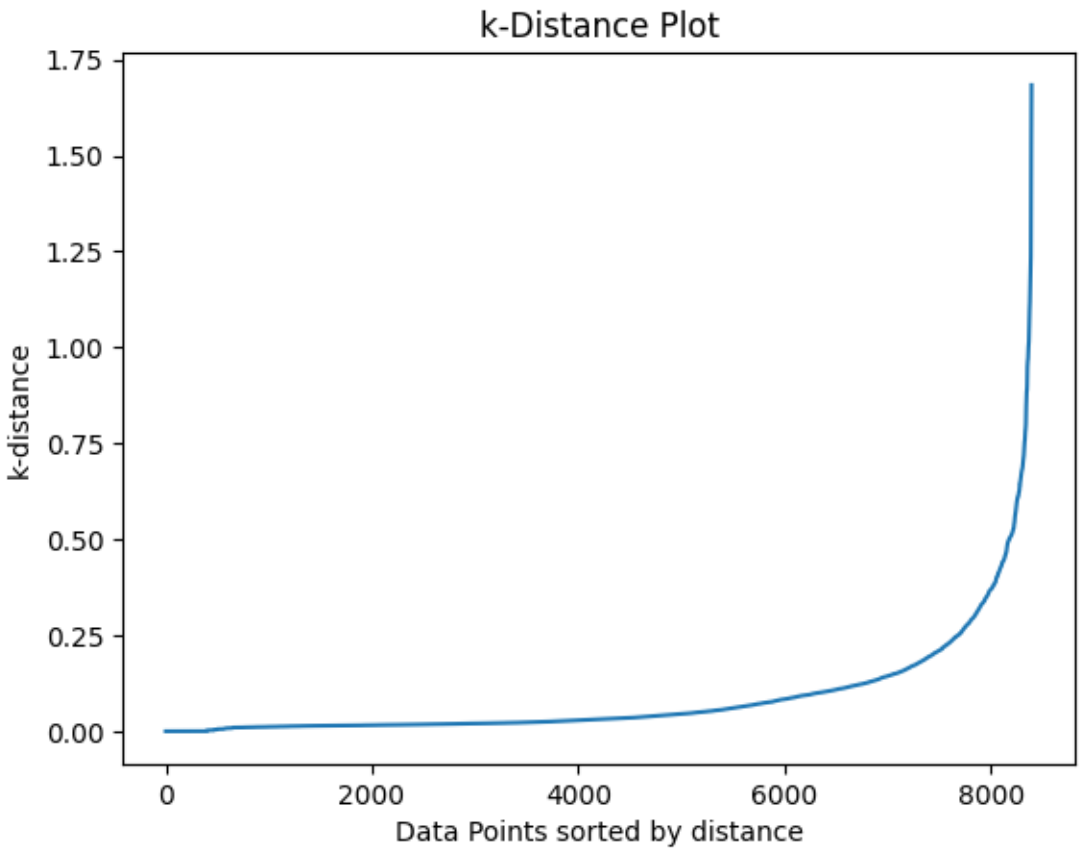
Distance-Based Clustering: DBSCAN relies on distance calculations to determine the neighborhood of each point. If your features have vastly different scales (e.g., one feature ranges from 0 to 1, while another ranges from 0 to 10,000), the clustering result might be dominated by the feature with the larger range. Standardizing your data ensures that each feature contributes equally to the distance calculations.

[ ]:
```
import pandas as pd
import numpy as np

minmax_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv')
standard_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /
 ↪Assignment/standard_sampling.csv')
```

[ ]:
```
pip install dbscan
```

```
Requirement already satisfied: dbscan in /usr/local/lib/python3.10/dist-packages
(0.0.12)
Requirement already satisfied: numpy<2,>=1.13.3 in
/usr/local/lib/python3.10/dist-packages (from dbscan) (1.26.4)
```

[ ]:
```
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt

# Fit Nearest Neighbors
neighbors = NearestNeighbors(n_neighbors=5)
neighbors_fit = neighbors.fit(minmax_data)
distances, indices = neighbors_fit.kneighbors(minmax_data)

# Sort and plot the distances
distances = np.sort(distances[:, 4], axis=0)  # 4th nearest neighbor
plt.plot(distances)
plt.ylabel('k-distance')
plt.xlabel('Data Points sorted by distance')
plt.title('k-Distance Plot')
plt.show()
```



[ ]:
```
from sklearn.cluster import DBSCAN

selected_vars = ['count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']

minmax_select = minmax_data[selected_vars]
standard_select = standard_data[selected_vars]
```

[ ]:
```
from sklearn.cluster import DBSCAN

# Define the DBSCAN model with the chosen parameters
dbscan_model = DBSCAN(eps=0.8, min_samples=20)  # Example parameters; adjust as␣
 ↪needed
dbscan_model_standard = DBSCAN(eps=2.5, min_samples=20)
```

```
# Fit DBSCAN on the MinMax scaled data
dbscan_minmax_labels = dbscan_model.fit_predict(minmax_select)

# Fit DBSCAN on the Standard scaled data
dbscan_standard_labels = dbscan_model_standard.fit_predict(standard_select)

# Calculate the number of clusters found (excluding noise) for MinMax data
n_clusters_minmax = len(set(dbscan_minmax_labels)) - (1 if -1 in␣
 ↪dbscan_minmax_labels else 0)
print(f"Estimated number of clusters (MinMax): {n_clusters_minmax}")

# Calculate the number of clusters found (excluding noise) for Standard data
n_clusters_standard = len(set(dbscan_standard_labels)) - (1 if -1 in␣
 ↪dbscan_standard_labels else 0)
print(f"Estimated number of clusters (Standard): {n_clusters_standard}")
```

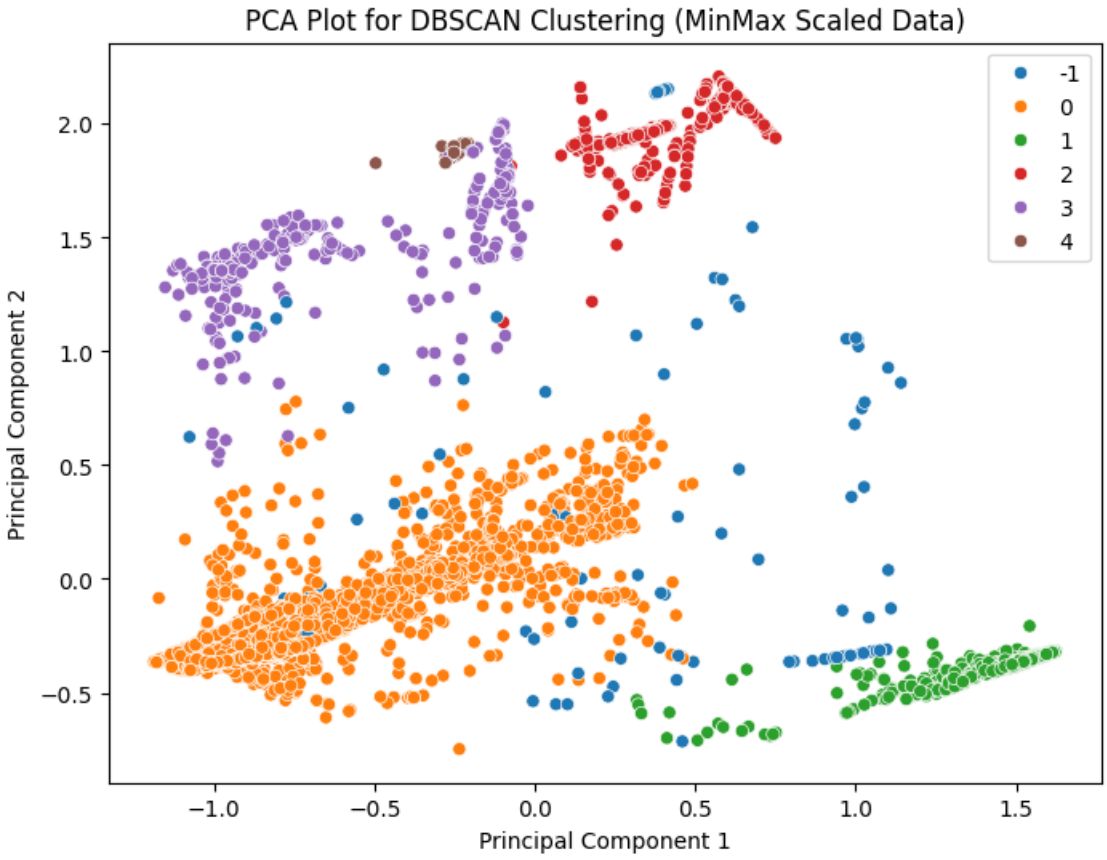
```
Estimated number of clusters (MinMax): 5
Estimated number of clusters (Standard): 5
```

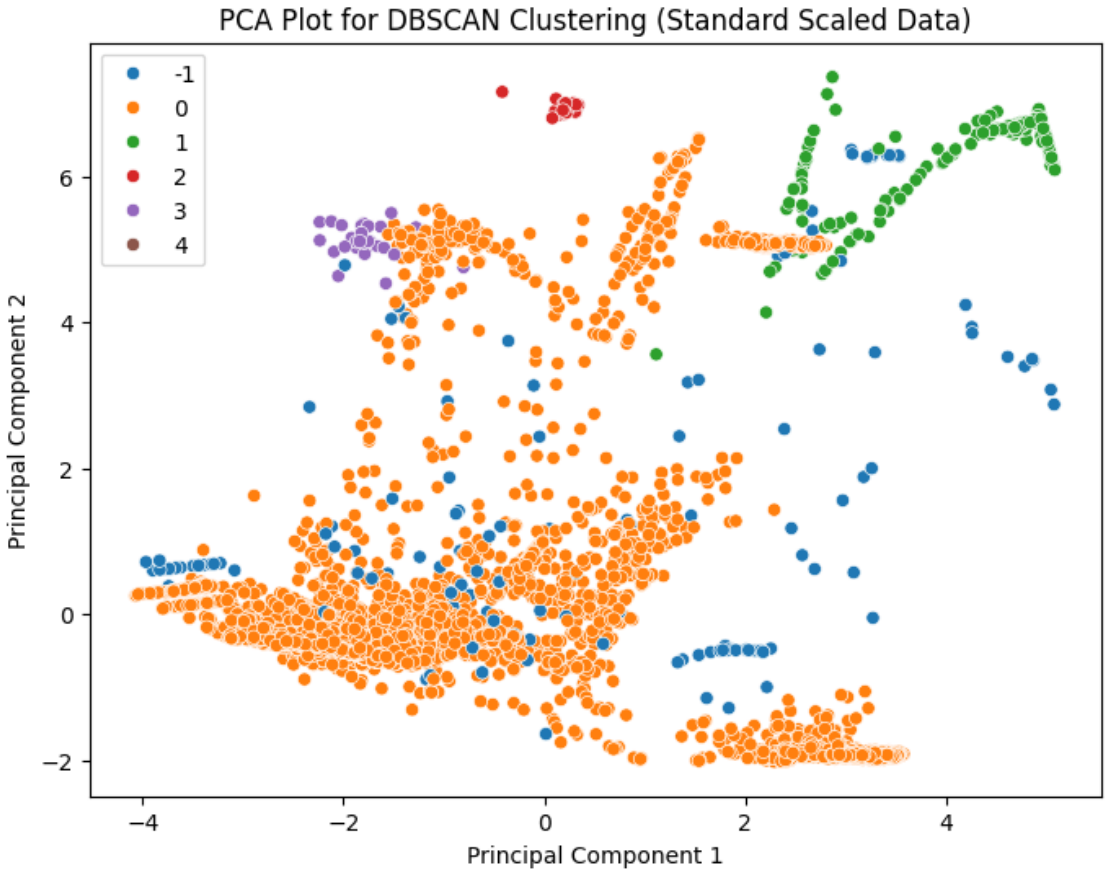
[ ]:
```
from sklearn.metrics import silhouette_score

# Calculate silhouette score for MinMax data
if n_clusters_minmax > 1:  # Silhouette score is not defined for a single␣
 ↪cluster
    silhouette_avg_minmax = silhouette_score(minmax_data, dbscan_minmax_labels)
    print(f"Silhouette Score (MinMax): {silhouette_avg_minmax}")
else:
    print("Silhouette score is not defined for a single cluster (MinMax).")

# Calculate silhouette score for Standard data
if n_clusters_standard > 1:  # Silhouette score is not defined for a single␣
 ↪cluster
    silhouette_avg_standard = silhouette_score(standard_data,␣
 ↪dbscan_standard_labels)
    print(f"Silhouette Score (Standard): {silhouette_avg_standard}")
else:
    print("Silhouette score is not defined for a single cluster (Standard).")
```

```
Silhouette Score (MinMax): 0.5740262202109881
Silhouette Score (Standard): 0.36121880999644335
```

[ ]:
```
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt

# Apply PCA for MinMax scaled data
```

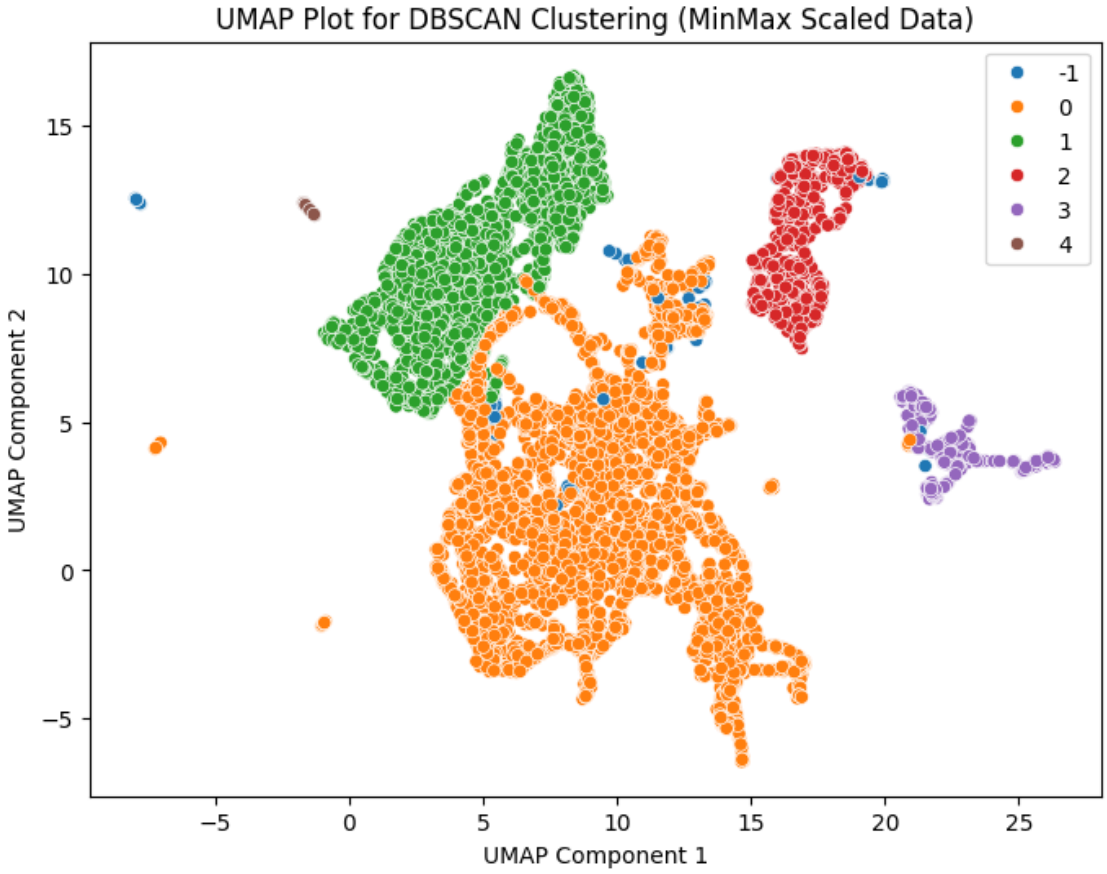
```
pca_minmax = PCA(n_components=2)
X_pca_minmax = pca_minmax.fit_transform(minmax_data)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca_minmax[:, 0], y=X_pca_minmax[:, 1],␣
 ↪hue=dbscan_minmax_labels, palette='tab10')
plt.title('PCA Plot for DBSCAN Clustering (MinMax Scaled Data)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Apply PCA for Standard scaled data
pca_standard = PCA(n_components=2)
X_pca_standard = pca_standard.fit_transform(standard_data)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca_standard[:, 0], y=X_pca_standard[:, 1],␣
 ↪hue=dbscan_standard_labels, palette='tab10')
plt.title('PCA Plot for DBSCAN Clustering (Standard Scaled Data)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

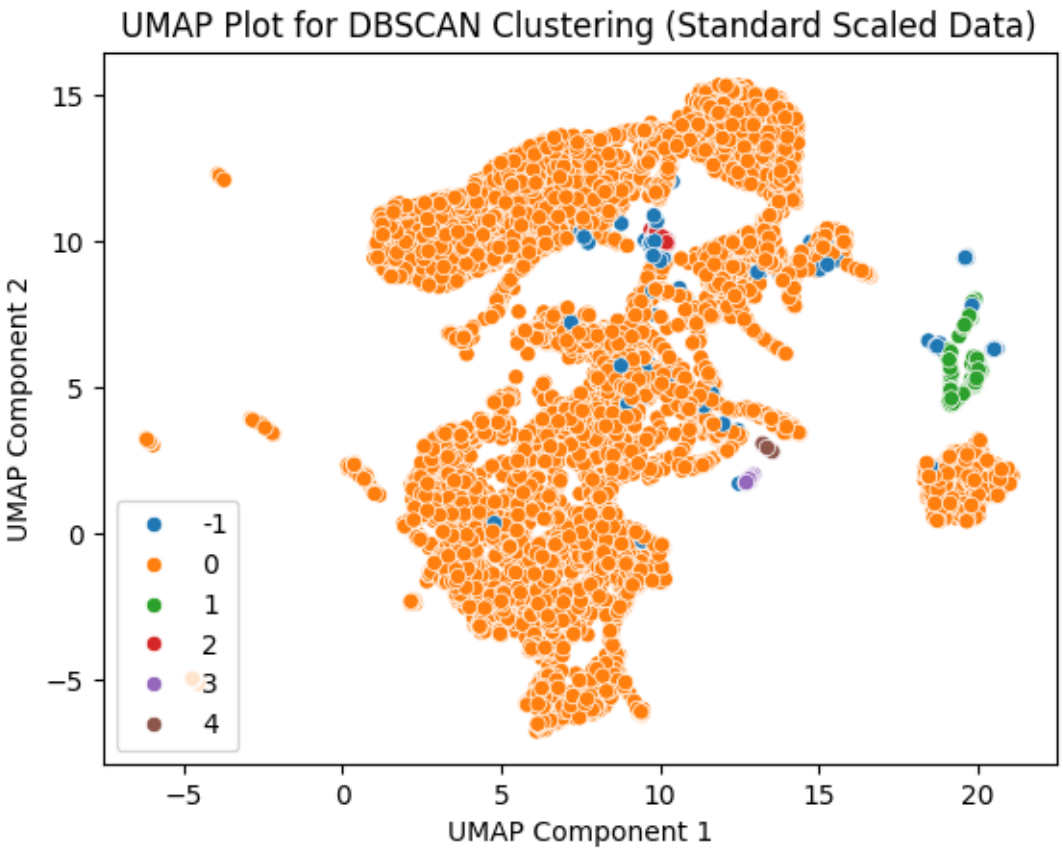




[ ]:
```
import umap
import seaborn as sns
import matplotlib.pyplot as plt

# UMAP plot for MinMax scaled data
umap_reducer_minmax = umap.UMAP(n_neighbors=15, min_dist=0.5, n_components=3)
X_umap_minmax = umap_reducer_minmax.fit_transform(minmax_select)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_umap_minmax[:, 0], y=X_umap_minmax[:, 1],␣
 ↪hue=dbscan_minmax_labels, palette='tab10')
plt.title('UMAP Plot for DBSCAN Clustering (MinMax Scaled Data)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.show()

# UMAP plot for Standard scaled data
```

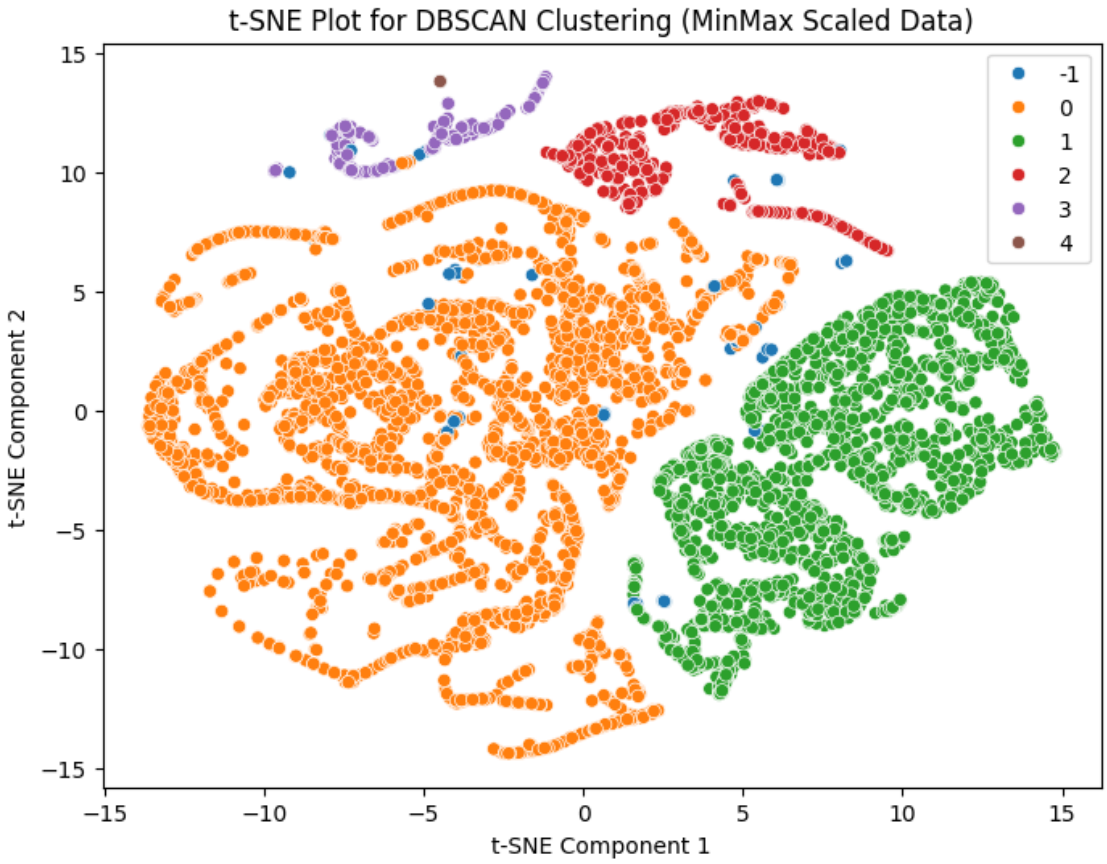
```
umap_reducer_standard = umap.UMAP(n_neighbors=15, min_dist=0.5, n_components=3)
X_umap_standard = umap_reducer_standard.fit_transform(standard_select)

sns.scatterplot(x=X_umap_standard[:, 0], y=X_umap_standard[:, 1],␣
 ↪hue=dbscan_standard_labels, palette='tab10')
plt.title('UMAP Plot for DBSCAN Clustering (Standard Scaled Data)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.show()
```

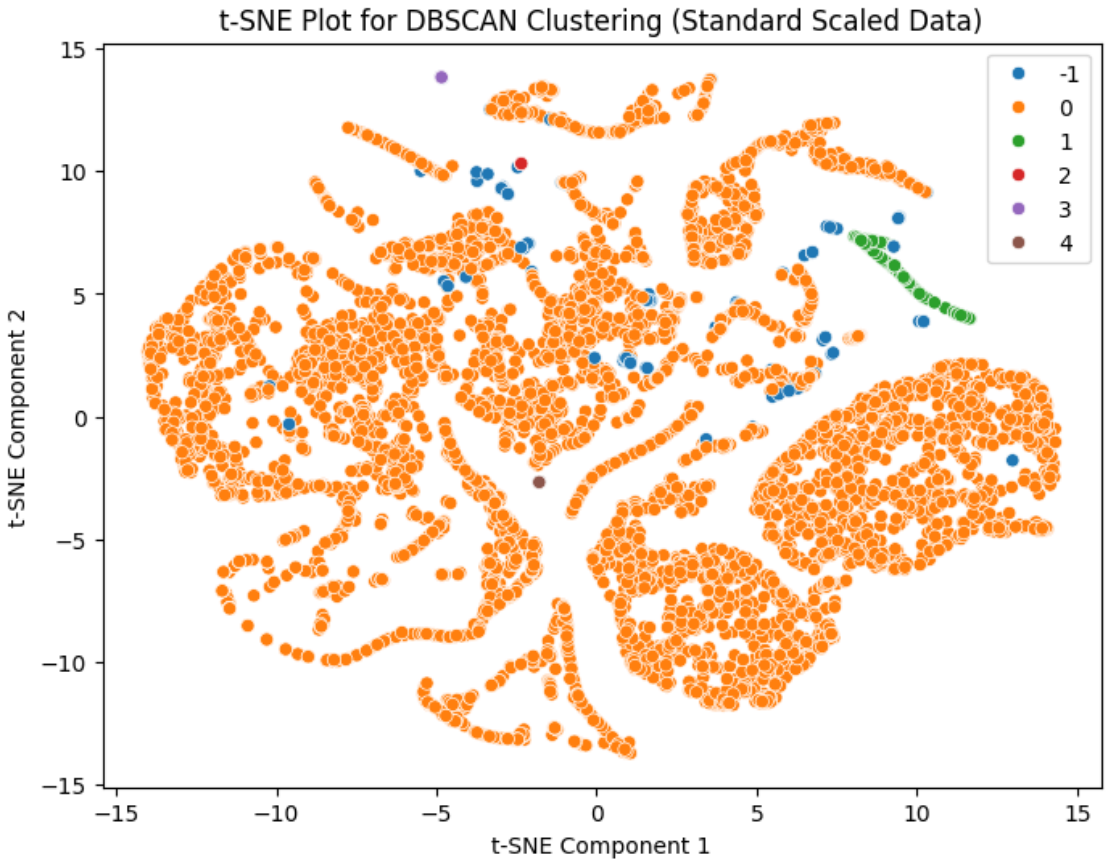




[ ]:
```
from sklearn.manifold import TSNE
import seaborn as sns
import matplotlib.pyplot as plt

# t-SNE plot for MinMax scaled data
tsne_minmax = TSNE(n_components=2, perplexity=30, n_iter=300, random_state=42)
X_tsne_minmax = tsne_minmax.fit_transform(minmax_select)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_tsne_minmax[:, 0], y=X_tsne_minmax[:, 1],␣
↪hue=dbscan_minmax_labels, palette='tab10')
plt.title('t-SNE Plot for DBSCAN Clustering (MinMax Scaled Data)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()

# t-SNE plot for Standard scaled data
tsne_standard = TSNE(n_components=2, perplexity=30, n_iter=300, random_state=42)
X_tsne_standard = tsne_standard.fit_transform(standard_select)
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_tsne_standard[:, 0], y=X_tsne_standard[:, 1],␣
 ↪hue=dbscan_standard_labels, palette='tab10')
plt.title('t-SNE Plot for DBSCAN Clustering (Standard Scaled Data)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```





6.3.1 6.3.1 Fine Tuning DBSCAN

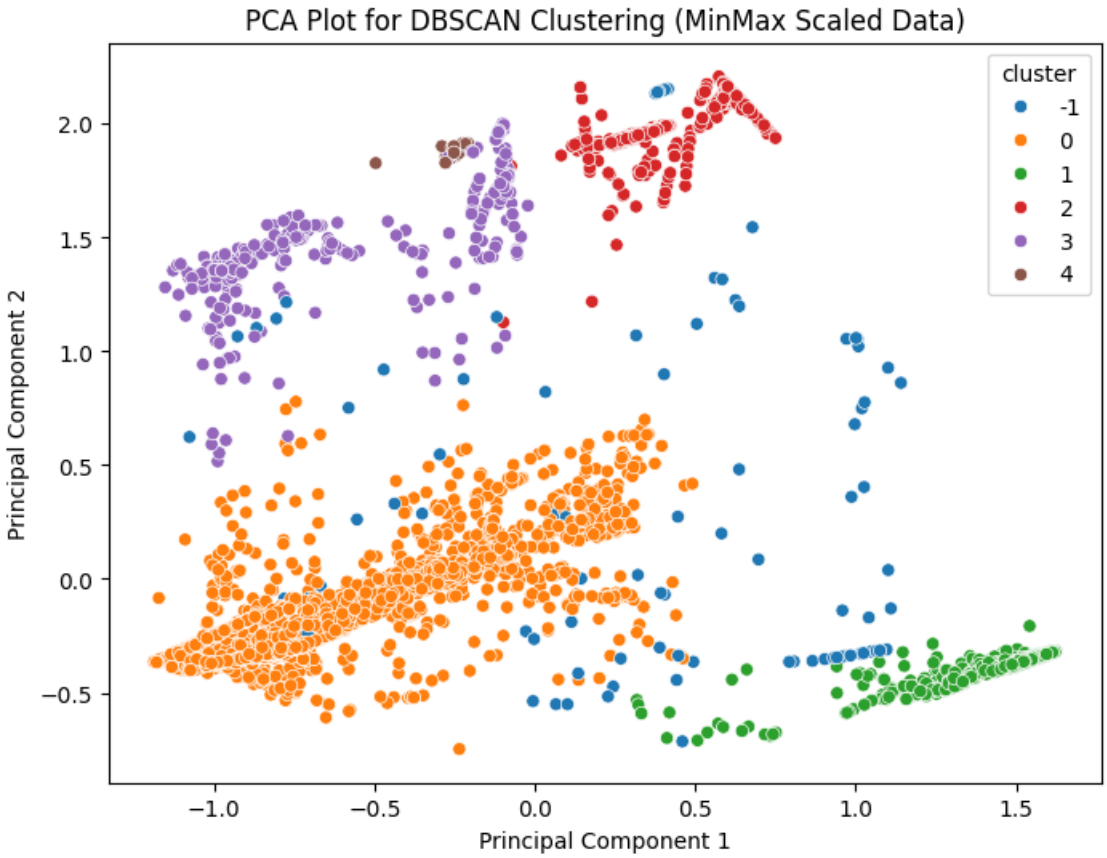
[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
import seaborn as sns
import umap
from sklearn.manifold import TSNE

# Load the data
minmax_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
  ↪minmax_sampling.csv')
standard_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /
  ↪Assignment/standard_sampling.csv')

# Define selected variables
```

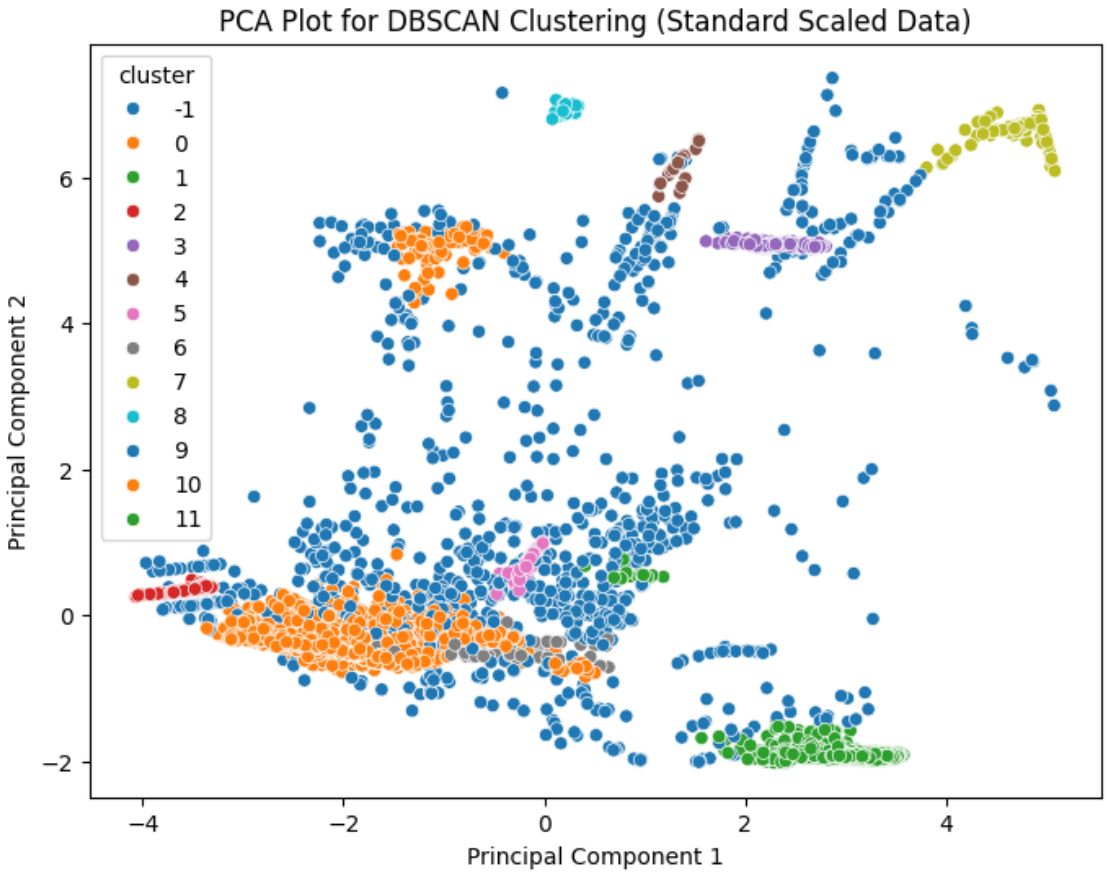
```
selected_vars = ['count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']

# Subset the data
X_minmax = minmax_data[selected_vars]
X_standard = standard_data[selected_vars]

# Function to fine-tune DBSCAN
def fine_tune_dbscan(X, dataset_name):
    best_eps = 0.8
    best_min_samples = 20
    best_silhouette = -1
    eps_values = np.arange(0.5, 1.5, 0.1)
    min_samples_values = range(5, 25, 5)

    for eps in eps_values:
        for min_samples in min_samples_values:
            dbscan_model = DBSCAN(eps=eps, min_samples=min_samples).fit(X)
            labels = dbscan_model.labels_
            n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

            if n_clusters > 1:  # Only calculate silhouette score if there is␣
 ↪more than one cluster
                silhouette_avg = silhouette_score(X, labels)
                if silhouette_avg > best_silhouette:
                    best_silhouette = silhouette_avg
                    best_eps = eps
                    best_min_samples = min_samples

    print(f"Best parameters for {dataset_name}: eps={best_eps},␣
 ↪min_samples={best_min_samples}")
    print(f"Best Silhouette Score for {dataset_name}: {best_silhouette}")
    return best_eps, best_min_samples

# Fine-tune DBSCAN for MinMax scaled data
best_eps_minmax, best_min_samples_minmax = fine_tune_dbscan(X_minmax, "MinMax␣
 ↪Scaled Data")

# Fine-tune DBSCAN for Standard scaled data
best_eps_standard, best_min_samples_standard = fine_tune_dbscan(X_standard,␣
 ↪"Standard Scaled Data")
```

```
# Apply DBSCAN with best parameters for MinMax scaled data
dbscan_best_model_minmax = DBSCAN(eps=best_eps_minmax,␣
 ↪min_samples=best_min_samples_minmax).fit(X_minmax)
minmax_data['cluster'] = dbscan_best_model_minmax.labels_

# Apply DBSCAN with best parameters for Standard scaled data
dbscan_best_model_standard = DBSCAN(eps=best_eps_standard,␣
 ↪min_samples=best_min_samples_standard).fit(X_standard)
standard_data['cluster'] = dbscan_best_model_standard.labels_
```

```
Best parameters for MinMax Scaled Data: eps=0.7999999999999999, min_samples=20
Best Silhouette Score for MinMax Scaled Data: 0.5740262202109881
Best parameters for Standard Scaled Data: eps=0.9999999999999999, min_samples=20
Best Silhouette Score for Standard Scaled Data: 0.47473430552843193
```

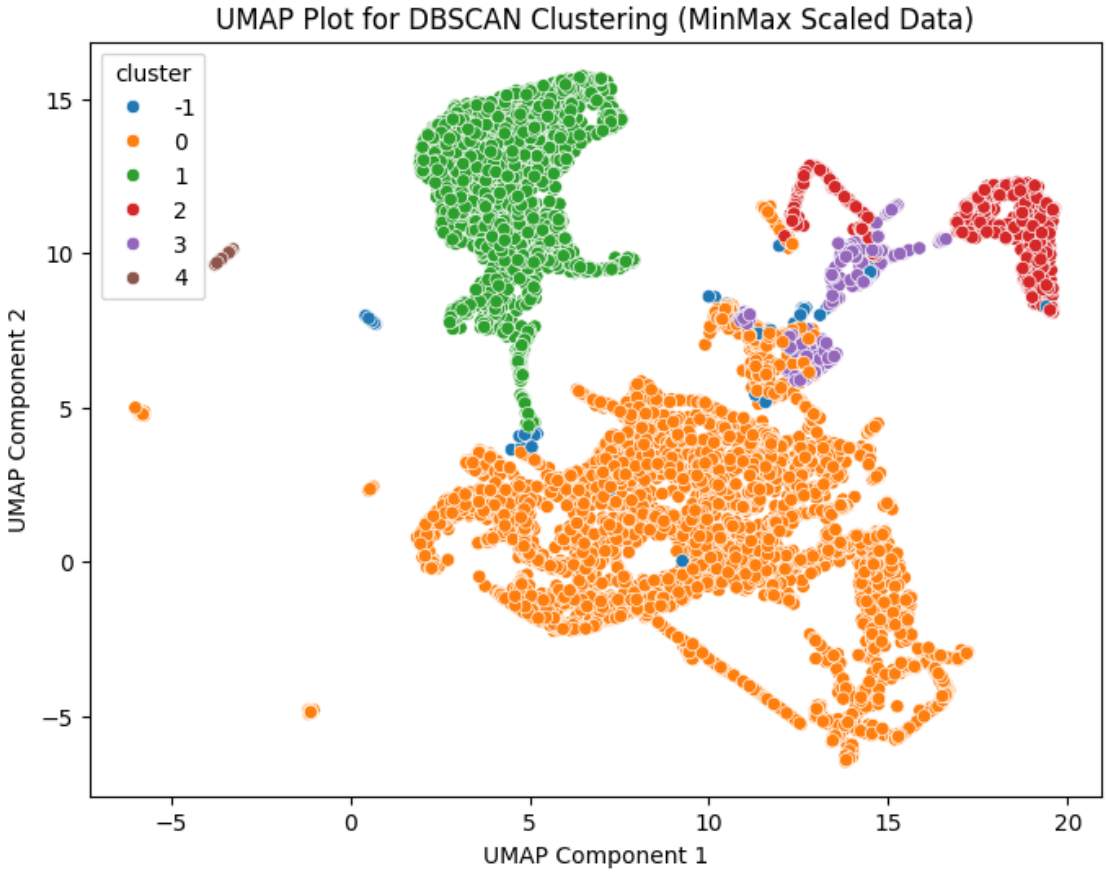
[ ]:
```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Apply PCA for MinMax scaled data
pca_minmax = PCA(n_components=2)
X_pca_minmax = pca_minmax.fit_transform(X_minmax)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca_minmax[:, 0], y=X_pca_minmax[:, 1],␣
 ↪hue=minmax_data['cluster'], palette='tab10')
plt.title('PCA Plot for DBSCAN Clustering (MinMax Scaled Data)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

# Apply PCA for Standard scaled data
pca_standard = PCA(n_components=2)
X_pca_standard = pca_standard.fit_transform(X_standard)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca_standard[:, 0], y=X_pca_standard[:, 1],␣
 ↪hue=standard_data['cluster'], palette='tab10')
plt.title('PCA Plot for DBSCAN Clustering (Standard Scaled Data)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

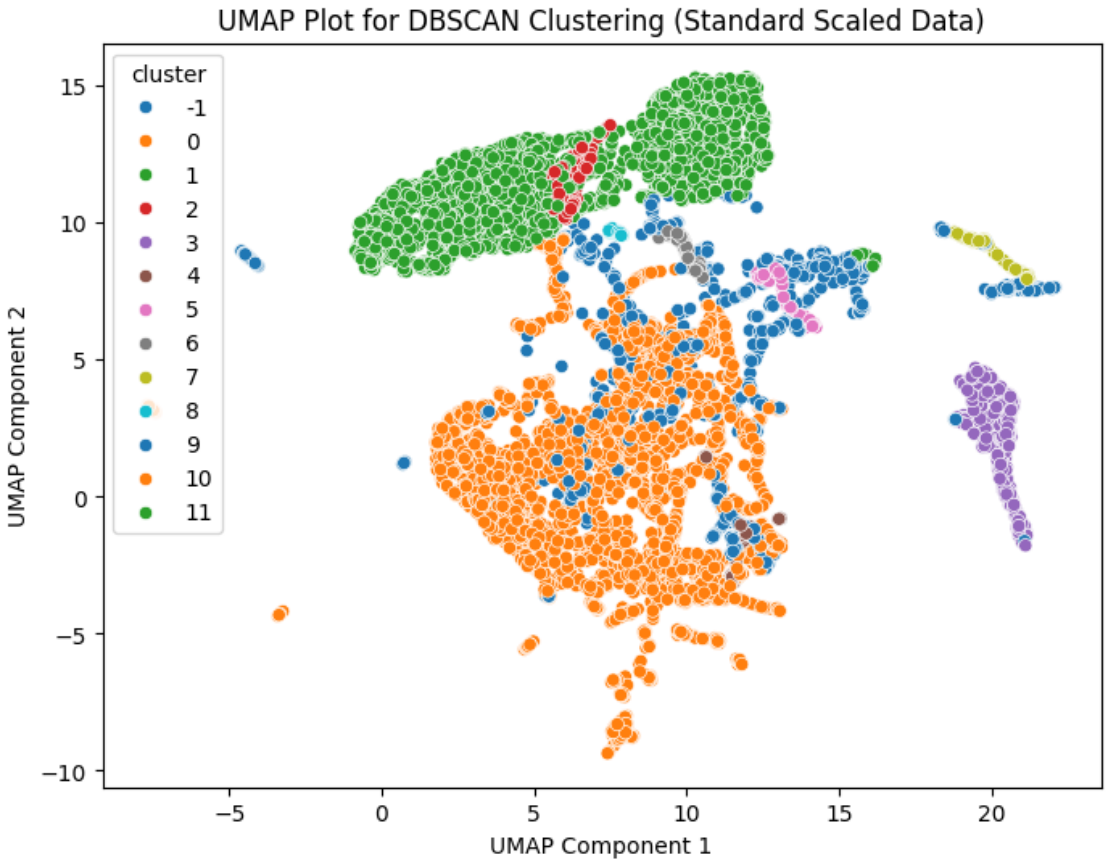

PCA Plot for DBSCAN Clustering (MinMax Scaled Data)
cluster
-1
0
1
2
3
4
Principal Component 2
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Principal Component 1



[ ]:
```
import umap
import matplotlib.pyplot as plt
import seaborn as sns

# UMAP plot for MinMax scaled data
umap_reducer_minmax = umap.UMAP(n_neighbors=15, min_dist=0.5, n_components=3)
X_umap_minmax = umap_reducer_minmax.fit_transform(X_minmax)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_umap_minmax[:, 0], y=X_umap_minmax[:, 1],␣
 ↪hue=minmax_data['cluster'], palette='tab10')
plt.title('UMAP Plot for DBSCAN Clustering (MinMax Scaled Data)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.show()

# UMAP plot for Standard scaled data
umap_reducer_standard = umap.UMAP(n_neighbors=15, min_dist=0.5, n_components=3)
X_umap_standard = umap_reducer_standard.fit_transform(X_standard)
```

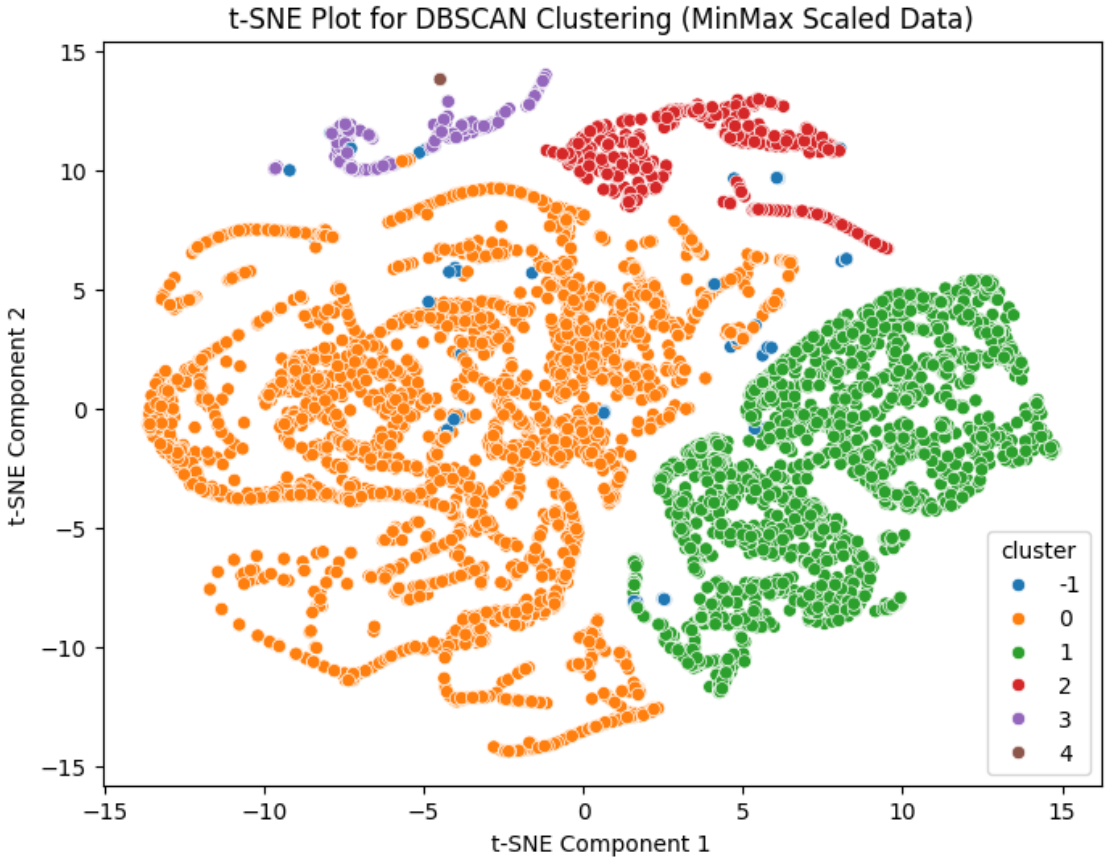
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_umap_standard[:, 0], y=X_umap_standard[:, 1],␣
 ↪hue=standard_data['cluster'], palette='tab10')
plt.title('UMAP Plot for DBSCAN Clustering (Standard Scaled Data)')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.show()
```

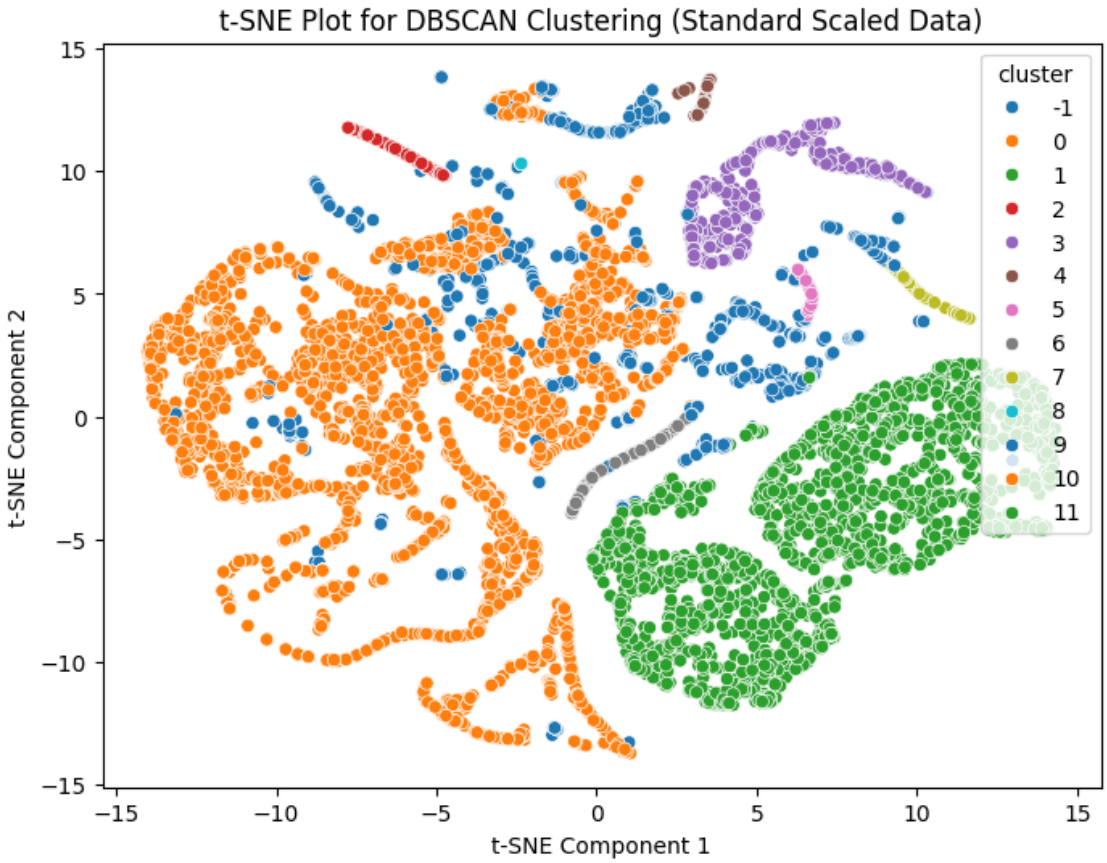




[ ]:
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import seaborn as sns

# t-SNE plot for MinMax scaled data
tsne_minmax = TSNE(n_components=2, perplexity=30, n_iter=300, random_state=42)
X_tsne_minmax = tsne_minmax.fit_transform(X_minmax)

plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_tsne_minmax[:, 0], y=X_tsne_minmax[:, 1],␣
↪hue=minmax_data['cluster'], palette='tab10')
plt.title('t-SNE Plot for DBSCAN Clustering (MinMax Scaled Data)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()

# t-SNE plot for Standard scaled data
tsne_standard = TSNE(n_components=2, perplexity=30, n_iter=300, random_state=42)
X_tsne_standard = tsne_standard.fit_transform(X_standard)
```

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_tsne_standard[:, 0], y=X_tsne_standard[:, 1],␣
 ↪hue=standard_data['cluster'], palette='tab10')
plt.title('t-SNE Plot for DBSCAN Clustering (Standard Scaled Data)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```





6.3.2 6.3.2 Evaluation after tuning

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score, calinski_harabasz_score,␣
 ↪davies_bouldin_score
from sklearn.model_selection import train_test_split

# Load the data
minmax_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv')
standard_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /
 ↪Assignment/standard_sampling.csv')

# Define selected variables
selected_vars = ['count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
                 'num_compromised', 'num_root', 'num_file_creations',␣
 ↪'num_shells',
```

```
                'num_access_files', 'num_outbound_cmds', 'total_bytes',␣
 ↪'serror_rate',
                'rerror_rate', 'srv_rerror_rate', 'same_srv_rate',␣
 ↪'diff_srv_rate',
                'srv_diff_host_rate', 'dst_host_same_srv_rate',␣
 ↪'dst_host_diff_srv_rate',
                'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
                'dst_host_serror_rate', 'dst_host_srv_serror_rate',
                'dst_host_rerror_rate', 'dst_host_srv_rerror_rate']

# Subset the data
minmax_select = minmax_data[selected_vars]
standard_select = standard_data[selected_vars]

# Convert to numpy arrays
X_minmax = minmax_select.to_numpy()
X_standard = standard_select.to_numpy()

# Function to fine-tune DBSCAN and calculate evaluation metrics
def evaluate_dbscan(X, dataset_name):
    best_eps = 0.8
    best_min_samples = 20
    best_silhouette = -1
    eps_values = np.arange(0.5, 1.5, 0.1)
    min_samples_values = range(5, 25, 5)

    # Fine-tune DBSCAN
    for eps in eps_values:
        for min_samples in min_samples_values:
            dbscan_model = DBSCAN(eps=eps, min_samples=min_samples).fit(X)
            labels = dbscan_model.labels_
            n_clusters = len(set(labels)) - (1 if -1 in labels else 0)

            if n_clusters > 1:  # Only calculate silhouette score if there is␣
 ↪more than one cluster
                silhouette_avg = silhouette_score(X, labels)
                if silhouette_avg > best_silhouette:
                    best_silhouette = silhouette_avg
                    best_eps = eps
                    best_min_samples = min_samples

    # Fit DBSCAN with the best parameters
    dbscan_best_model = DBSCAN(eps=best_eps, min_samples=best_min_samples).
 ↪fit(X)
    labels = dbscan_best_model.labels_
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
```

```
    # Calculate metrics
    if n_clusters > 1:
        silhouette_avg = silhouette_score(X, labels)
        calinski_harabasz = calinski_harabasz_score(X, labels)
        davies_bouldin = davies_bouldin_score(X, labels)
    else:
        silhouette_avg = -1
        calinski_harabasz = -1
        davies_bouldin = -1

    print(f"Evaluation Results for {dataset_name}:")
    print(f"Best Parameters: eps={best_eps}, min_samples={best_min_samples}")
    print(f"Silhouette Score: {silhouette_avg}")
    print(f"Calinski-Harabasz Score: {calinski_harabasz}")
    print(f"Davies-Bouldin Score: {davies_bouldin}\n")

    return best_eps, best_min_samples, silhouette_avg, calinski_harabasz,␣
 ↪davies_bouldin

# Evaluate DBSCAN for MinMax scaled data
eps_minmax, min_samples_minmax, silhouette_minmax, calinski_minmax,␣
 ↪davies_minmax = evaluate_dbscan(X_minmax, "MinMax Scaled Data")

# Evaluate DBSCAN for Standard scaled data
eps_standard, min_samples_standard, silhouette_standard, calinski_standard,␣
 ↪davies_standard = evaluate_dbscan(X_standard, "Standard Scaled Data")
```

```
Evaluation Results for MinMax Scaled Data:
Best Parameters: eps=0.7999999999999999, min_samples=20
Silhouette Score: 0.5740262202109881
Calinski-Harabasz Score: 3874.49309829715
Davies-Bouldin Score: 1.408205394283434

Evaluation Results for Standard Scaled Data:
Best Parameters: eps=0.9999999999999999, min_samples=20
Silhouette Score: 0.47473430552843193
Calinski-Harabasz Score: 802.618391534531
Davies-Bouldin Score: 1.269965313455729
```

6.3.3 6.3.3 Summary of Comparison

Conclusion:

[ ]:
```
# Compare the results
print("Comparison of DBSCAN Clustering Performance:")
```

```
print(f"MinMax Scaled Data: Silhouette Score = {silhouette_minmax},␣
 ↪Calinski-Harabasz Score = {calinski_minmax}, Davies-Bouldin Score =␣
 ↪{davies_minmax}")
print(f"Standard Scaled Data: Silhouette Score = {silhouette_standard},␣
 ↪Calinski-Harabasz Score = {calinski_standard}, Davies-Bouldin Score =␣
 ↪{davies_standard}")

if silhouette_minmax > silhouette_standard:
    print("MinMax Scaled Data provides a higher silhouette score.")
else:
    print("Standard Scaled Data provides a higher silhouette score.")

if calinski_minmax > calinski_standard:
    print("MinMax Scaled Data provides better-defined clusters (higher␣
 ↪Calinski-Harabasz score).")
else:
    print("Standard Scaled Data provides better-defined clusters.")

if davies_minmax < davies_standard:
    print("MinMax Scaled Data provides better clustering (lower Davies-Bouldin␣
 ↪score).")
else:
    print("Standard Scaled Data provides better clustering.")
```

```
Comparison of DBSCAN Clustering Performance:
MinMax Scaled Data: Silhouette Score = 0.5740262202109881, Calinski-Harabasz
Score = 3874.49309829715, Davies-Bouldin Score = 1.408205394283434
Standard Scaled Data: Silhouette Score = 0.47473430552843193, Calinski-Harabasz
Score = 802.618391534531, Davies-Bouldin Score = 1.269965313455729
MinMax Scaled Data provides a higher silhouette score.
MinMax Scaled Data provides better-defined clusters (higher Calinski-Harabasz
score).
Standard Scaled Data provides better clustering.
```

6.4 6.4 Autoencoder

An autoencoder is a type of neural network used for unsupervised learning tasks, particularly for tasks like dimensionality reduction, feature learning, or anomaly detection. The goal of an autoencoder is to learn a compressed, lower-dimensional representation of the input data (called the latent space or bottleneck) and then reconstruct the original input from this compressed representation.

Autoencoders are widely used in various applications like image compression, denoising, anomaly detection, and more.

Structure of an Autoencoder: 1. Encoder: - The encoder compresses the input data into a lower-dimensional latent representation. It does this by progressively reducing the dimensionality through a series of layers. - The encoder transforms the input into a hidden representation (the bottleneck) via a mapping function = ( ).

2. Latent Space (Bottleneck):

- This is the compressed, lower-dimensional representation of the data, which captures the essential features or patterns in the data. The size of this bottleneck controls the amount of compression and feature abstraction.

3. Decoder:

- The decoder reconstructs the input from the compressed representation. It tries to reverse the encoding process and recreate the original input as closely as possible.
- The decoder transforms the latent representation back to the original input space via a mapping function = ( ), where is the reconstruction of the input .

4. Reconstruction Loss

- The autoencoder is trained by minimizing the reconstruction loss, which is a measure of the difference between the original input and the reconstructed output. This loss is usually measured using Mean Squared Error (MSE) or other similar metrics.

6.4.1 6.4.1 Fine Tuning Autoencoder

[ ]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-
packages (2.17.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.11.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (24.1)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
```

```
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (71.0.4)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.64.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.0)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow) (3.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in
/usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.44.0)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (13.8.1)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages
(from keras>=3.2.0->tensorflow) (0.12.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorflow) (3.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.18,>=2.17->tensorflow) (3.7)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.18,>=2.17->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from
```

```
tensorboard<2.18,>=2.17->tensorflow) (3.0.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.18,>=2.17->tensorflow) (2.1.5)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow)
(2.16.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-
packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.0->tensorflow) (0.1.2)
```

[ ]:
```python
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Load the datasets
minmax_df = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv')
standard_df = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪standard_sampling.csv')

# Select the specified variables
variables = [
    'count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'
]

minmax_data = minmax_df[variables].values
standard_data = standard_df[variables].values
```

[ ]:
```python
import tensorflow as tf
from tensorflow.keras import layers

# Define the autoencoder model
input_dim = len(variables)
encoding_dim = 16  # Latent space dimensionality

input_layer = layers.Input(shape=(input_dim,))
encoder = layers.Dense(encoding_dim, activation="relu")(input_layer)
decoder = layers.Dense(input_dim, activation="sigmoid")(encoder)
```

```
autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train the autoencoder on the minmax scaled data
autoencoder.fit(minmax_data, minmax_data, epochs=50, batch_size=32,␣
 ↪shuffle=True)

# Encode the data using the trained encoder
minmax_encoded = tf.keras.Model(inputs=input_layer, outputs=encoder).
 ↪predict(minmax_data)
standard_encoded = tf.keras.Model(inputs=input_layer, outputs=encoder).
 ↪predict(standard_data)
```

```
Epoch 1/50
263/263             2s 2ms/step -
loss: 0.1815
Epoch 2/50
263/263             1s 2ms/step -
loss: 0.0305
Epoch 3/50
263/263             2s 4ms/step -
loss: 0.0164
Epoch 4/50
263/263             1s 5ms/step -
loss: 0.0116
Epoch 5/50
263/263             3s 7ms/step -
loss: 0.0088
Epoch 6/50
263/263             2s 6ms/step -
loss: 0.0065
Epoch 7/50
263/263             2s 4ms/step -
loss: 0.0053
Epoch 8/50
263/263             2s 6ms/step -
loss: 0.0043
Epoch 9/50
263/263             2s 3ms/step -
loss: 0.0037
Epoch 10/50
263/263             2s 5ms/step -
loss: 0.0033
Epoch 11/50
263/263             3s 5ms/step -
loss: 0.0028
```

```
Epoch 12/50
263/263            3s 6ms/step -
loss: 0.0025
Epoch 13/50
263/263            2s 7ms/step -
loss: 0.0023
Epoch 14/50
263/263            2s 5ms/step -
loss: 0.0020
Epoch 15/50
263/263            2s 5ms/step -
loss: 0.0020
Epoch 16/50
263/263            1s 4ms/step -
loss: 0.0018
Epoch 17/50
263/263            1s 4ms/step -
loss: 0.0017
Epoch 18/50
263/263            1s 4ms/step -
loss: 0.0015
Epoch 19/50
263/263            1s 4ms/step -
loss: 0.0015
Epoch 20/50
263/263            1s 4ms/step -
loss: 0.0014
Epoch 21/50
263/263            1s 4ms/step -
loss: 0.0013
Epoch 22/50
263/263            2s 6ms/step -
loss: 0.0012
Epoch 23/50
263/263            1s 5ms/step -
loss: 0.0011
Epoch 24/50
263/263            2s 2ms/step -
loss: 0.0010
Epoch 25/50
263/263            1s 2ms/step -
loss: 9.9386e-04
Epoch 26/50
263/263            0s 2ms/step -
loss: 9.3437e-04
Epoch 27/50
263/263            0s 1ms/step -
loss: 9.4460e-04
```

```
Epoch 28/50
263/263            1s 2ms/step -
loss: 9.0007e-04
Epoch 29/50
263/263            1s 2ms/step -
loss: 8.8239e-04
Epoch 30/50
263/263            1s 2ms/step -
loss: 8.4058e-04
Epoch 31/50
263/263            0s 1ms/step -
loss: 7.8789e-04
Epoch 32/50
263/263            0s 2ms/step -
loss: 7.6792e-04
Epoch 33/50
263/263            1s 2ms/step -
loss: 7.5825e-04
Epoch 34/50
263/263            0s 2ms/step -
loss: 7.3814e-04
Epoch 35/50
263/263            1s 2ms/step -
loss: 7.3505e-04
Epoch 36/50
263/263            0s 2ms/step -
loss: 7.4536e-04
Epoch 37/50
263/263            1s 2ms/step -
loss: 7.2967e-04
Epoch 38/50
263/263            0s 2ms/step -
loss: 6.8985e-04
Epoch 39/50
263/263            1s 2ms/step -
loss: 6.6702e-04
Epoch 40/50
263/263            1s 2ms/step -
loss: 6.8763e-04
Epoch 41/50
263/263            1s 2ms/step -
loss: 6.8542e-04
Epoch 42/50
263/263            1s 3ms/step -
loss: 6.5240e-04
Epoch 43/50
263/263            1s 3ms/step -
loss: 6.4715e-04
```

```
Epoch 44/50
263/263             1s 3ms/step -
loss: 6.3784e-04
Epoch 45/50
263/263             1s 2ms/step -
loss: 6.3693e-04
Epoch 46/50
263/263             0s 2ms/step -
loss: 6.3961e-04
Epoch 47/50
263/263             1s 2ms/step -
loss: 6.1622e-04
Epoch 48/50
263/263             0s 2ms/step -
loss: 5.9954e-04
Epoch 49/50
263/263             1s 2ms/step -
loss: 6.1359e-04
Epoch 50/50
263/263             1s 2ms/step -
loss: 6.0947e-04
263/263             0s 1ms/step
263/263             0s 1ms/step
```

[ ]:
```python
import tensorflow as tf
from tensorflow.keras import layers

# Define the autoencoder model
input_dim = len(variables)
encoding_dim = 16  # Latent space dimensionality

input_layer = layers.Input(shape=(input_dim,))
encoder = layers.Dense(encoding_dim, activation="relu")(input_layer)
decoder = layers.Dense(input_dim, activation="sigmoid")(encoder)

autoencoder = tf.keras.Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train the autoencoder on the minmax scaled data
autoencoder.fit(minmax_data, minmax_data, epochs=50, batch_size=32,␣
 ↪shuffle=True)

# Encode the data using the trained encoder
minmax_encoded = tf.keras.Model(inputs=input_layer, outputs=encoder).
 ↪predict(minmax_data)
standard_encoded = tf.keras.Model(inputs=input_layer, outputs=encoder).
 ↪predict(standard_data)
```

```
Epoch 1/50
263/263            2s 2ms/step -
loss: 0.1687
Epoch 2/50
263/263            2s 4ms/step -
loss: 0.0325
Epoch 3/50
263/263            1s 4ms/step -
loss: 0.0145
Epoch 4/50
263/263            2s 6ms/step -
loss: 0.0091
Epoch 5/50
263/263            3s 7ms/step -
loss: 0.0066
Epoch 6/50
263/263            2s 4ms/step -
loss: 0.0048
Epoch 7/50
263/263            1s 3ms/step -
loss: 0.0038
Epoch 8/50
263/263            1s 4ms/step -
loss: 0.0034
Epoch 9/50
263/263            2s 3ms/step -
loss: 0.0029
Epoch 10/50
263/263            2s 5ms/step -
loss: 0.0026
Epoch 11/50
263/263            2s 4ms/step -
loss: 0.0025
Epoch 12/50
263/263            1s 4ms/step -
loss: 0.0020
Epoch 13/50
263/263            2s 7ms/step -
loss: 0.0019
Epoch 14/50
263/263            2s 6ms/step -
loss: 0.0017
Epoch 15/50
263/263            2s 3ms/step -
loss: 0.0016
Epoch 16/50
263/263            1s 3ms/step -
loss: 0.0015
```

```
Epoch 17/50
263/263            1s 3ms/step -
loss: 0.0014
Epoch 18/50
263/263            2s 5ms/step -
loss: 0.0013
Epoch 19/50
263/263            2s 4ms/step -
loss: 0.0012
Epoch 20/50
263/263            1s 4ms/step -
loss: 0.0013
Epoch 21/50
263/263            2s 6ms/step -
loss: 0.0011
Epoch 22/50
263/263            3s 6ms/step -
loss: 0.0011
Epoch 23/50
263/263            1s 2ms/step -
loss: 0.0011
Epoch 24/50
263/263            1s 1ms/step -
loss: 0.0010
Epoch 25/50
263/263            0s 2ms/step -
loss: 9.2842e-04
Epoch 26/50
263/263            1s 1ms/step -
loss: 9.3720e-04
Epoch 27/50
263/263            0s 2ms/step -
loss: 9.5601e-04
Epoch 28/50
263/263            1s 1ms/step -
loss: 9.2293e-04
Epoch 29/50
263/263            0s 2ms/step -
loss: 8.0395e-04
Epoch 30/50
263/263            1s 2ms/step -
loss: 7.9096e-04
Epoch 31/50
263/263            1s 2ms/step -
loss: 7.8209e-04
Epoch 32/50
263/263            0s 2ms/step -
loss: 7.6547e-04
```

```
Epoch 33/50
263/263            1s 1ms/step -
loss: 7.6212e-04
Epoch 34/50
263/263            1s 1ms/step -
loss: 7.2796e-04
Epoch 35/50
263/263            1s 1ms/step -
loss: 7.3751e-04
Epoch 36/50
263/263            1s 1ms/step -
loss: 7.3171e-04
Epoch 37/50
263/263            0s 1ms/step -
loss: 6.6538e-04
Epoch 38/50
263/263            0s 2ms/step -
loss: 6.7693e-04
Epoch 39/50
263/263            1s 1ms/step -
loss: 6.7344e-04
Epoch 40/50
263/263            1s 1ms/step -
loss: 6.9832e-04
Epoch 41/50
263/263            0s 2ms/step -
loss: 6.3679e-04
Epoch 42/50
263/263            1s 3ms/step -
loss: 6.3278e-04
Epoch 43/50
263/263            1s 3ms/step -
loss: 6.1867e-04
Epoch 44/50
263/263            1s 3ms/step -
loss: 6.0432e-04
Epoch 45/50
263/263            1s 2ms/step -
loss: 5.8923e-04
Epoch 46/50
263/263            1s 2ms/step -
loss: 6.0160e-04
Epoch 47/50
263/263            0s 2ms/step -
loss: 5.8812e-04
Epoch 48/50
263/263            1s 1ms/step -
loss: 5.6868e-04
```

```
Epoch 49/50
263/263             1s 1ms/step -
loss: 5.7570e-04
Epoch 50/50
263/263             0s 1ms/step -
loss: 5.8256e-04
263/263             0s 1ms/step
263/263             0s 1ms/step
```

[ ]:
```python
from tensorflow.keras.callbacks import EarlyStopping

# Set up an early stopping criterion to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
 ↪restore_best_weights=True)

# Train the autoencoder on MinMax Scaled Data (Fine-Tuning)
autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),␣
 ↪loss='mse')
history_minmax = autoencoder.fit(minmax_data, minmax_data, epochs=150,␣
 ↪batch_size=32,
                                 validation_split=0.2, shuffle=True,␣
 ↪callbacks=[early_stopping])

# Train the autoencoder on Standard Scaled Data (Fine-Tuning)
autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),␣
 ↪loss='mse')
history_standard = autoencoder.fit(standard_data, standard_data, epochs=150,␣
 ↪batch_size=32,
                                   validation_split=0.2, shuffle=True,␣
 ↪callbacks=[early_stopping])
```

```
Epoch 1/150
210/210             1s 2ms/step -
loss: 5.5782e-04 - val_loss: 5.5638e-04
Epoch 2/150
210/210             1s 2ms/step -
loss: 5.4412e-04 - val_loss: 5.5544e-04
Epoch 3/150
210/210             0s 2ms/step -
loss: 5.6382e-04 - val_loss: 5.5299e-04
Epoch 4/150
210/210             1s 2ms/step -
loss: 5.5491e-04 - val_loss: 5.5211e-04
Epoch 5/150
210/210             0s 2ms/step -
loss: 5.4674e-04 - val_loss: 5.5253e-04
Epoch 6/150
```

```
210/210             1s 2ms/step -
loss: 5.6061e-04 - val_loss: 5.6047e-04
Epoch 7/150
210/210             0s 2ms/step -
loss: 5.6234e-04 - val_loss: 5.4673e-04
Epoch 8/150
210/210             1s 2ms/step -
loss: 5.3442e-04 - val_loss: 5.4750e-04
Epoch 9/150
210/210             0s 2ms/step -
loss: 5.4362e-04 - val_loss: 5.4946e-04
Epoch 10/150
210/210             1s 4ms/step -
loss: 5.3812e-04 - val_loss: 5.5496e-04
Epoch 11/150
210/210             1s 3ms/step -
loss: 5.4472e-04 - val_loss: 5.5349e-04
Epoch 12/150
210/210             1s 4ms/step -
loss: 5.2786e-04 - val_loss: 5.5415e-04
Epoch 13/150
210/210             1s 3ms/step -
loss: 5.2352e-04 - val_loss: 5.4496e-04
Epoch 14/150
210/210             1s 2ms/step -
loss: 5.3493e-04 - val_loss: 5.4558e-04
Epoch 15/150
210/210             1s 2ms/step -
loss: 5.3081e-04 - val_loss: 5.3960e-04
Epoch 16/150
210/210             0s 2ms/step -
loss: 5.3408e-04 - val_loss: 5.5002e-04
Epoch 17/150
210/210             1s 2ms/step -
loss: 5.4304e-04 - val_loss: 5.3905e-04
Epoch 18/150
210/210             1s 2ms/step -
loss: 5.2767e-04 - val_loss: 5.4330e-04
Epoch 19/150
210/210             0s 2ms/step -
loss: 5.1761e-04 - val_loss: 5.4969e-04
Epoch 20/150
210/210             0s 2ms/step -
loss: 5.3079e-04 - val_loss: 5.3962e-04
Epoch 21/150
210/210             1s 2ms/step -
loss: 5.4986e-04 - val_loss: 5.3798e-04
Epoch 22/150
```

```
210/210             1s 2ms/step -
loss: 5.4012e-04 - val_loss: 5.4045e-04
Epoch 23/150
210/210             0s 2ms/step -
loss: 5.1314e-04 - val_loss: 5.4223e-04
Epoch 24/150
210/210             1s 2ms/step -
loss: 5.5953e-04 - val_loss: 5.3344e-04
Epoch 25/150
210/210             0s 2ms/step -
loss: 5.2745e-04 - val_loss: 5.3727e-04
Epoch 26/150
210/210             1s 2ms/step -
loss: 5.1500e-04 - val_loss: 5.4680e-04
Epoch 27/150
210/210             1s 2ms/step -
loss: 5.1496e-04 - val_loss: 5.3410e-04
Epoch 28/150
210/210             1s 2ms/step -
loss: 5.4454e-04 - val_loss: 5.3464e-04
Epoch 29/150
210/210             0s 2ms/step -
loss: 5.3312e-04 - val_loss: 5.3913e-04
Epoch 30/150
210/210             1s 2ms/step -
loss: 5.1815e-04 - val_loss: 5.3381e-04
Epoch 31/150
210/210             1s 2ms/step -
loss: 5.4039e-04 - val_loss: 5.4198e-04
Epoch 32/150
210/210             1s 3ms/step -
loss: 4.9145e-04 - val_loss: 5.4183e-04
Epoch 33/150
210/210             1s 4ms/step -
loss: 5.2637e-04 - val_loss: 5.3686e-04
Epoch 34/150
210/210             1s 4ms/step -
loss: 5.2431e-04 - val_loss: 5.3202e-04
Epoch 35/150
210/210             1s 2ms/step -
loss: 5.0929e-04 - val_loss: 5.3663e-04
Epoch 36/150
210/210             1s 2ms/step -
loss: 5.0815e-04 - val_loss: 5.3764e-04
Epoch 37/150
210/210             0s 2ms/step -
loss: 4.9644e-04 - val_loss: 5.2875e-04
Epoch 38/150
```

```
210/210            1s 2ms/step -
loss: 5.3295e-04 - val_loss: 5.2989e-04
Epoch 39/150
210/210            0s 2ms/step -
loss: 4.9621e-04 - val_loss: 5.4501e-04
Epoch 40/150
210/210            1s 2ms/step -
loss: 4.9410e-04 - val_loss: 5.2974e-04
Epoch 41/150
210/210            0s 2ms/step -
loss: 5.2253e-04 - val_loss: 5.3730e-04
Epoch 42/150
210/210            1s 2ms/step -
loss: 5.0592e-04 - val_loss: 5.2795e-04
Epoch 43/150
210/210            0s 2ms/step -
loss: 5.0041e-04 - val_loss: 5.3707e-04
Epoch 44/150
210/210            0s 2ms/step -
loss: 5.1170e-04 - val_loss: 5.3005e-04
Epoch 45/150
210/210            0s 2ms/step -
loss: 5.1749e-04 - val_loss: 5.2756e-04
Epoch 46/150
210/210            1s 2ms/step -
loss: 5.0481e-04 - val_loss: 5.2503e-04
Epoch 47/150
210/210            0s 2ms/step -
loss: 4.9967e-04 - val_loss: 5.2515e-04
Epoch 48/150
210/210            1s 2ms/step -
loss: 5.0468e-04 - val_loss: 5.2494e-04
Epoch 49/150
210/210            1s 2ms/step -
loss: 5.0191e-04 - val_loss: 5.2754e-04
Epoch 50/150
210/210            1s 2ms/step -
loss: 5.0041e-04 - val_loss: 5.2004e-04
Epoch 51/150
210/210            0s 2ms/step -
loss: 4.9878e-04 - val_loss: 5.2223e-04
Epoch 52/150
210/210            0s 2ms/step -
loss: 4.8626e-04 - val_loss: 5.2867e-04
Epoch 53/150
210/210            0s 2ms/step -
loss: 5.0282e-04 - val_loss: 5.2247e-04
Epoch 54/150
```

```
210/210            1s 3ms/step -
loss: 5.1170e-04 - val_loss: 5.2403e-04
Epoch 55/150
210/210            1s 3ms/step -
loss: 5.1399e-04 - val_loss: 5.2107e-04
Epoch 56/150
210/210            1s 4ms/step -
loss: 5.0126e-04 - val_loss: 5.2136e-04
Epoch 57/150
210/210            1s 2ms/step -
loss: 5.3788e-04 - val_loss: 5.1503e-04
Epoch 58/150
210/210            1s 2ms/step -
loss: 4.9134e-04 - val_loss: 5.1869e-04
Epoch 59/150
210/210            1s 2ms/step -
loss: 4.8832e-04 - val_loss: 5.2311e-04
Epoch 60/150
210/210            1s 2ms/step -
loss: 5.1527e-04 - val_loss: 5.2043e-04
Epoch 61/150
210/210            1s 2ms/step -
loss: 4.9101e-04 - val_loss: 5.1575e-04
Epoch 62/150
210/210            1s 2ms/step -
loss: 5.0101e-04 - val_loss: 5.1365e-04
Epoch 63/150
210/210            1s 2ms/step -
loss: 4.9404e-04 - val_loss: 5.1039e-04
Epoch 64/150
210/210            1s 2ms/step -
loss: 4.9098e-04 - val_loss: 5.1503e-04
Epoch 65/150
210/210            0s 2ms/step -
loss: 4.8642e-04 - val_loss: 5.1011e-04
Epoch 66/150
210/210            1s 2ms/step -
loss: 4.9784e-04 - val_loss: 5.0615e-04
Epoch 67/150
210/210            1s 2ms/step -
loss: 4.9417e-04 - val_loss: 5.0971e-04
Epoch 68/150
210/210            1s 2ms/step -
loss: 4.9393e-04 - val_loss: 5.0969e-04
Epoch 69/150
210/210            1s 2ms/step -
loss: 4.7023e-04 - val_loss: 5.0753e-04
Epoch 70/150
```

```
210/210            1s 2ms/step -
loss: 4.8713e-04 - val_loss: 4.9899e-04
Epoch 71/150
210/210            0s 2ms/step -
loss: 4.9600e-04 - val_loss: 4.9850e-04
Epoch 72/150
210/210            1s 2ms/step -
loss: 5.0107e-04 - val_loss: 4.9876e-04
Epoch 73/150
210/210            1s 2ms/step -
loss: 4.7618e-04 - val_loss: 4.9825e-04
Epoch 74/150
210/210            1s 3ms/step -
loss: 4.8444e-04 - val_loss: 4.9294e-04
Epoch 75/150
210/210            1s 3ms/step -
loss: 4.8461e-04 - val_loss: 4.9036e-04
Epoch 76/150
210/210            1s 3ms/step -
loss: 5.0330e-04 - val_loss: 4.8745e-04
Epoch 77/150
210/210            1s 2ms/step -
loss: 4.9754e-04 - val_loss: 4.8525e-04
Epoch 78/150
210/210            1s 2ms/step -
loss: 4.8600e-04 - val_loss: 4.8643e-04
Epoch 79/150
210/210            0s 2ms/step -
loss: 4.8883e-04 - val_loss: 4.8621e-04
Epoch 80/150
210/210            1s 2ms/step -
loss: 4.8153e-04 - val_loss: 4.8665e-04
Epoch 81/150
210/210            0s 2ms/step -
loss: 4.8579e-04 - val_loss: 4.8193e-04
Epoch 82/150
210/210            1s 2ms/step -
loss: 4.8770e-04 - val_loss: 4.7959e-04
Epoch 83/150
210/210            0s 2ms/step -
loss: 4.8684e-04 - val_loss: 4.8918e-04
Epoch 84/150
210/210            1s 2ms/step -
loss: 4.7607e-04 - val_loss: 4.8040e-04
Epoch 85/150
210/210            0s 2ms/step -
loss: 4.7868e-04 - val_loss: 4.7995e-04
Epoch 86/150
```

```
210/210             1s 2ms/step -
loss: 4.8051e-04 - val_loss: 4.8224e-04
Epoch 87/150
210/210             1s 2ms/step -
loss: 4.7130e-04 - val_loss: 4.8223e-04
Epoch 88/150
210/210             0s 2ms/step -
loss: 4.6531e-04 - val_loss: 4.8407e-04
Epoch 89/150
210/210             1s 2ms/step -
loss: 4.8492e-04 - val_loss: 4.8450e-04
Epoch 90/150
210/210             1s 2ms/step -
loss: 4.7055e-04 - val_loss: 4.7883e-04
Epoch 91/150
210/210             1s 2ms/step -
loss: 4.7485e-04 - val_loss: 4.7952e-04
Epoch 92/150
210/210             1s 2ms/step -
loss: 4.8328e-04 - val_loss: 4.8032e-04
Epoch 93/150
210/210             1s 2ms/step -
loss: 4.7660e-04 - val_loss: 4.7363e-04
Epoch 94/150
210/210             1s 3ms/step -
loss: 4.5730e-04 - val_loss: 4.7712e-04
Epoch 95/150
210/210             1s 3ms/step -
loss: 4.7637e-04 - val_loss: 4.7965e-04
Epoch 96/150
210/210             3s 10ms/step -
loss: 4.6536e-04 - val_loss: 4.7802e-04
Epoch 97/150
210/210             1s 2ms/step -
loss: 4.6743e-04 - val_loss: 4.7607e-04
Epoch 98/150
210/210             1s 2ms/step -
loss: 4.6437e-04 - val_loss: 4.8129e-04
Epoch 99/150
210/210             1s 2ms/step -
loss: 4.8005e-04 - val_loss: 4.7690e-04
Epoch 100/150
210/210             0s 2ms/step -
loss: 4.6697e-04 - val_loss: 4.7237e-04
Epoch 101/150
210/210             1s 2ms/step -
loss: 4.6658e-04 - val_loss: 4.7496e-04
Epoch 102/150
```

```
210/210            0s 2ms/step -
loss: 4.9173e-04 - val_loss: 4.8306e-04
Epoch 103/150
210/210            1s 2ms/step -
loss: 4.6900e-04 - val_loss: 4.7372e-04
Epoch 104/150
210/210            1s 2ms/step -
loss: 4.6972e-04 - val_loss: 4.7576e-04
Epoch 105/150
210/210            1s 2ms/step -
loss: 4.9023e-04 - val_loss: 4.7338e-04
Epoch 106/150
210/210            1s 2ms/step -
loss: 4.7123e-04 - val_loss: 4.7654e-04
Epoch 107/150
210/210            1s 2ms/step -
loss: 4.6750e-04 - val_loss: 4.8219e-04
Epoch 108/150
210/210            1s 2ms/step -
loss: 4.9781e-04 - val_loss: 4.7401e-04
Epoch 109/150
210/210            1s 2ms/step -
loss: 4.6880e-04 - val_loss: 4.7766e-04
Epoch 110/150
210/210            1s 2ms/step -
loss: 4.7894e-04 - val_loss: 4.7506e-04
Epoch 1/150
210/210            2s 3ms/step -
loss: 0.5468 - val_loss: 0.5083
Epoch 2/150
210/210            1s 3ms/step -
loss: 0.6354 - val_loss: 0.5066
Epoch 3/150
210/210            1s 4ms/step -
loss: 0.5165 - val_loss: 0.5057
Epoch 4/150
210/210            1s 4ms/step -
loss: 0.9718 - val_loss: 0.5052
Epoch 5/150
210/210            1s 2ms/step -
loss: 0.4963 - val_loss: 0.5048
Epoch 6/150
210/210            0s 2ms/step -
loss: 0.8367 - val_loss: 0.5044
Epoch 7/150
210/210            1s 2ms/step -
loss: 0.5188 - val_loss: 0.5042
Epoch 8/150
```

```
210/210             0s 2ms/step -
loss: 0.5759 - val_loss: 0.5040
Epoch 9/150
210/210             1s 2ms/step -
loss: 0.6632 - val_loss: 0.5039
Epoch 10/150
210/210             1s 2ms/step -
loss: 0.7057 - val_loss: 0.5037
```

[ ]:
```python
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Perform K-Means clustering on the encoded data
kmeans_minmax = KMeans(n_clusters=3, random_state=42).fit(minmax_encoded)
kmeans_standard = KMeans(n_clusters=3, random_state=42).fit(standard_encoded)

# Calculate the silhouette score for clustering performance
silhouette_minmax = silhouette_score(minmax_encoded, kmeans_minmax.labels_)
silhouette_standard = silhouette_score(standard_encoded, kmeans_standard.
 ↪labels_)

print(f'Silhouette Score for MinMax Scaled Data: {silhouette_minmax}')
print(f'Silhouette Score for Standard Scaled Data: {silhouette_standard}')
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)

Silhouette Score for MinMax Scaled Data: 0.43209609389305115
Silhouette Score for Standard Scaled Data: 0.5216177105903625
```

###6.4.2 Evaluation after tuning

[ ]:
```python
from tensorflow.keras import Model

# Build the autoencoder model
input_dim = minmax_data.shape[1]  # Assuming you have already defined this
latent_dim = 16  # Example latent space size

# Encoder part
input_layer = tf.keras.layers.Input(shape=(input_dim,))
encoded = tf.keras.layers.Dense(64, activation='relu')(input_layer)
encoded = tf.keras.layers.Dense(32, activation='relu')(encoded)
latent = tf.keras.layers.Dense(latent_dim, activation='relu')(encoded)
```

```
# Decoder part
decoded = tf.keras.layers.Dense(32, activation='relu')(latent)
decoded = tf.keras.layers.Dense(64, activation='relu')(decoded)
output_layer = tf.keras.layers.Dense(input_dim, activation='sigmoid')(decoded)

# Complete autoencoder model
autoencoder = Model(input_layer, output_layer)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mse')

# Define the encoder model (input to latent space)
encoder = Model(inputs=input_layer, outputs=latent)

# Get the latent representations from the encoder (for MinMax Scaled Data)
minmax_encoded = encoder.predict(minmax_data)

# Get the latent representations from the encoder (for Standard Scaled Data)
standard_encoded = encoder.predict(standard_data)

# Apply K-Means on MinMax Encoded Data
kmeans_minmax = KMeans(n_clusters=3, random_state=42).fit(minmax_encoded)
silhouette_minmax = silhouette_score(minmax_encoded, kmeans_minmax.labels_)

# Apply K-Means on Standard Encoded Data
kmeans_standard = KMeans(n_clusters=3, random_state=42).fit(standard_encoded)
silhouette_standard = silhouette_score(standard_encoded, kmeans_standard.
 ↪labels_)

print(f'Silhouette Score for MinMax Scaled Data: {silhouette_minmax}')
print(f'Silhouette Score for Standard Scaled Data: {silhouette_standard}')
```

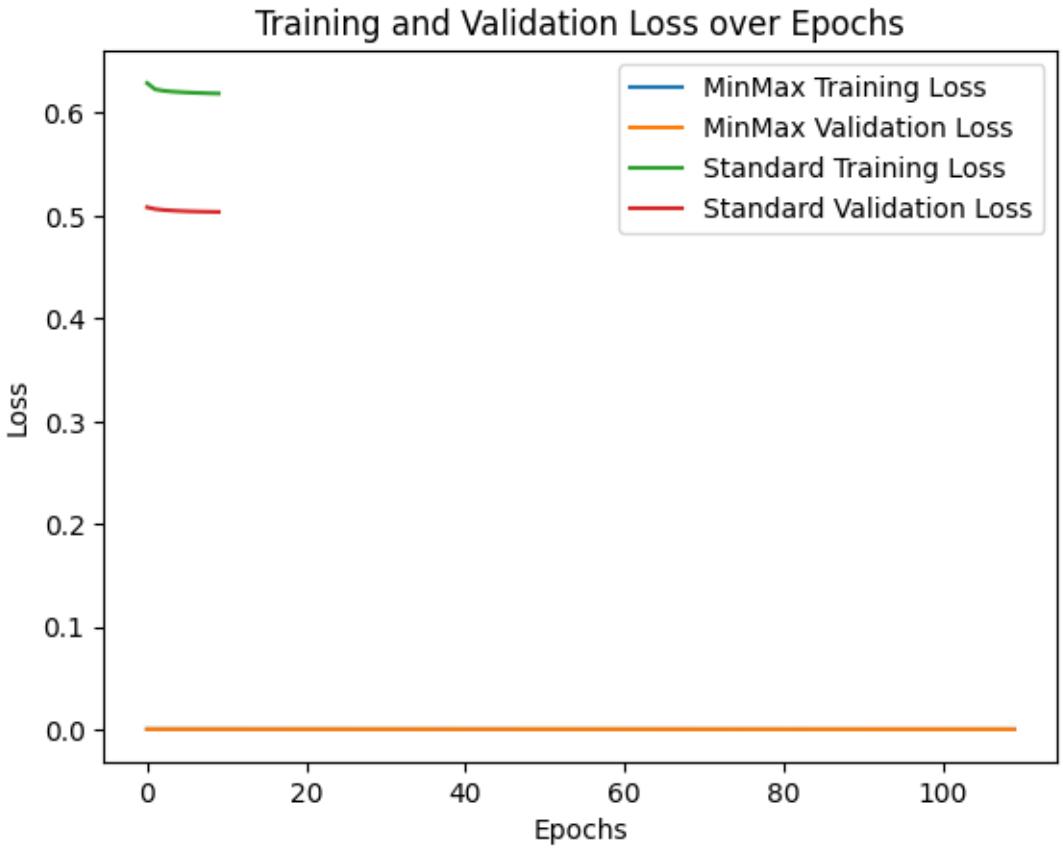
```
263/263             1s 3ms/step
263/263             1s 2ms/step
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
Silhouette Score for MinMax Scaled Data: 0.45855963230133057
Silhouette Score for Standard Scaled Data: 0.3897148668766022
```

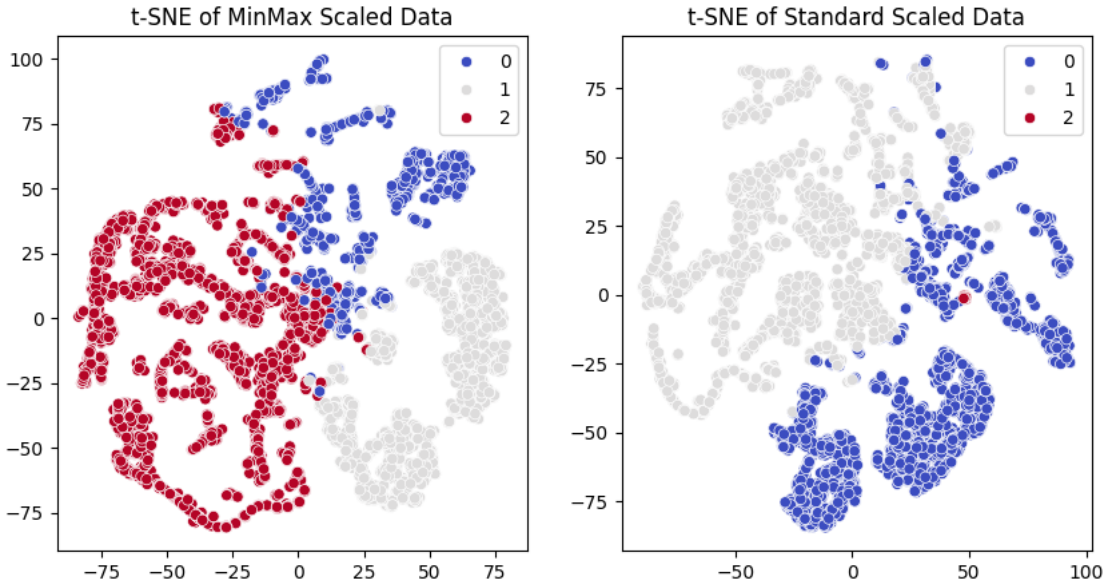
6.4.3 Summary of Comparison

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# Plot training & validation loss for MinMax Scaled Data
plt.plot(history_minmax.history['loss'], label='MinMax Training Loss')
plt.plot(history_minmax.history['val_loss'], label='MinMax Validation Loss')

# Plot training & validation loss for Standard Scaled Data
plt.plot(history_standard.history['loss'], label='Standard Training Loss')
plt.plot(history_standard.history['val_loss'], label='Standard Validation Loss')

plt.title('Training and Validation Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[ ]:
```
from sklearn.manifold import TSNE
import seaborn as sns
```

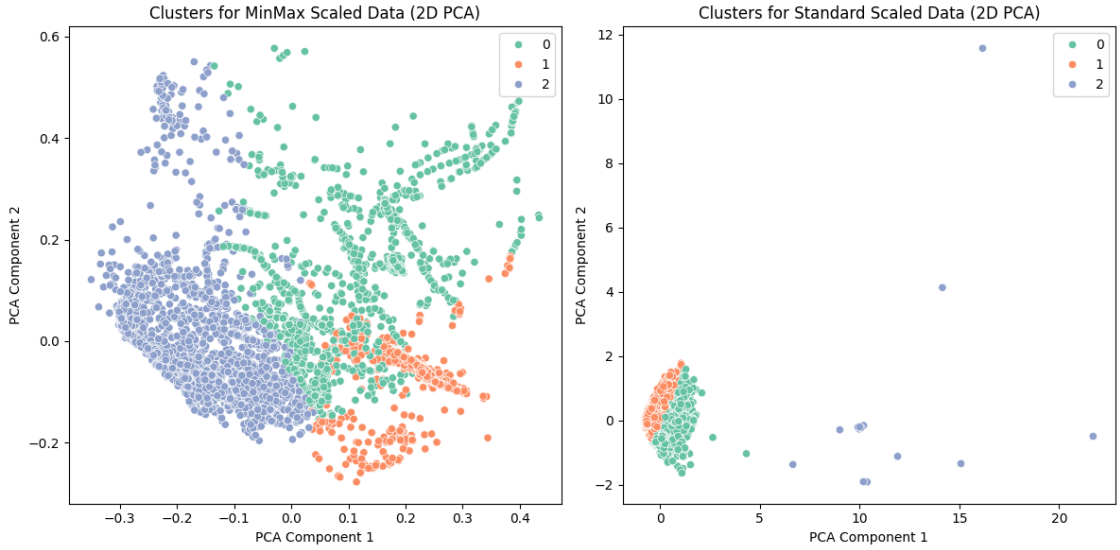
```
# Apply t-SNE on MinMax Encoded Data
tsne_minmax = TSNE(n_components=2).fit_transform(minmax_encoded)

# Apply t-SNE on Standard Encoded Data
tsne_standard = TSNE(n_components=2).fit_transform(standard_encoded)

# Plot the t-SNE results for MinMax Scaled Data
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(x=tsne_minmax[:, 0], y=tsne_minmax[:, 1], hue=kmeans_minmax.
 ↪labels_, palette='coolwarm')
plt.title('t-SNE of MinMax Scaled Data')

# Plot the t-SNE results for Standard Scaled Data
plt.subplot(1, 2, 2)
sns.scatterplot(x=tsne_standard[:, 0], y=tsne_standard[:, 1],␣
 ↪hue=kmeans_standard.labels_, palette='coolwarm')
plt.title('t-SNE of Standard Scaled Data')

plt.show()
```



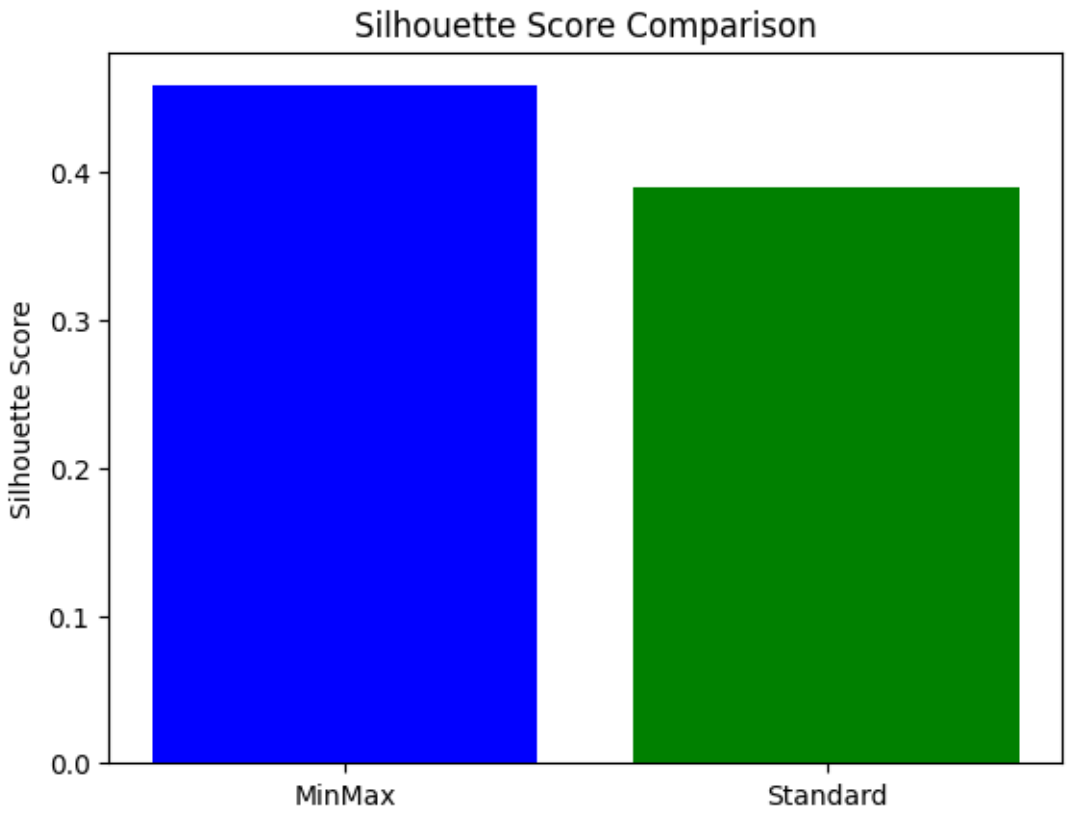
[ ]:
```
# Apply PCA to reduce the latent space to 2D
pca = PCA(n_components=2)
minmax_pca = pca.fit_transform(minmax_encoded)
standard_pca = pca.fit_transform(standard_encoded)

# Apply K-Means to get cluster labels
```

```
kmeans_minmax = KMeans(n_clusters=3, random_state=42).fit(minmax_encoded)
kmeans_standard = KMeans(n_clusters=3, random_state=42).fit(standard_encoded)

minmax_labels = kmeans_minmax.labels_
standard_labels = kmeans_standard.labels_

# Plot MinMax Scaled Data Clusters
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.scatterplot(x=minmax_pca[:, 0], y=minmax_pca[:, 1], hue=minmax_labels,␣
 ↪palette="Set2")
plt.title('Clusters for MinMax Scaled Data (2D PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

# Plot Standard Scaled Data Clusters
plt.subplot(1, 2, 2)
sns.scatterplot(x=standard_pca[:, 0], y=standard_pca[:, 1],␣
 ↪hue=standard_labels, palette="Set2")
plt.title('Clusters for Standard Scaled Data (2D PCA)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```



[ ]:
```
# Plotting Silhouette Scores for MinMax and Standard Scaled Data
silhouette_scores = [silhouette_minmax, silhouette_standard]
scaling_methods = ['MinMax', 'Standard']

plt.bar(scaling_methods, silhouette_scores, color=['blue', 'green'])
plt.title('Silhouette Score Comparison')
plt.ylabel('Silhouette Score')
plt.show()
```

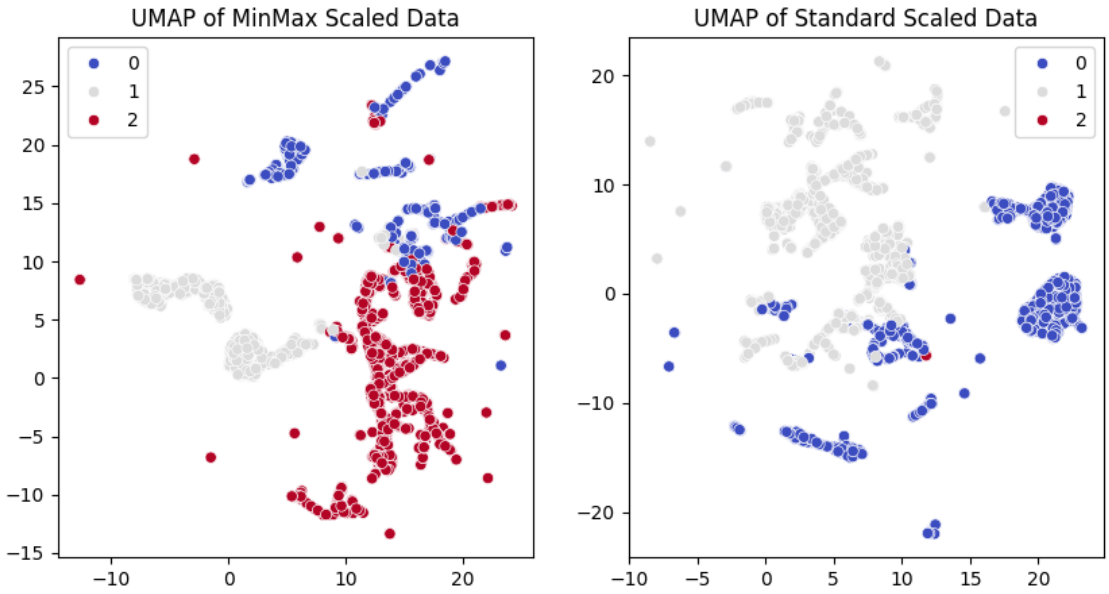


[ ]:
```
!pip install umap-learn

import umap.umap_ as umap

# Apply UMAP to MinMax Encoded Data
umap_minmax = umap.UMAP(n_components=2, random_state=42).
 ↪fit_transform(minmax_encoded)

# Apply UMAP to Standard Encoded Data
umap_standard = umap.UMAP(n_components=2, random_state=42).
 ↪fit_transform(standard_encoded)

# Plot UMAP results for MinMax Scaled Data
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(x=umap_minmax[:, 0], y=umap_minmax[:, 1], hue=kmeans_minmax.
 ↪labels_, palette='coolwarm')
plt.title('UMAP of MinMax Scaled Data')

# Plot UMAP results for Standard Scaled Data
plt.subplot(1, 2, 2)
```

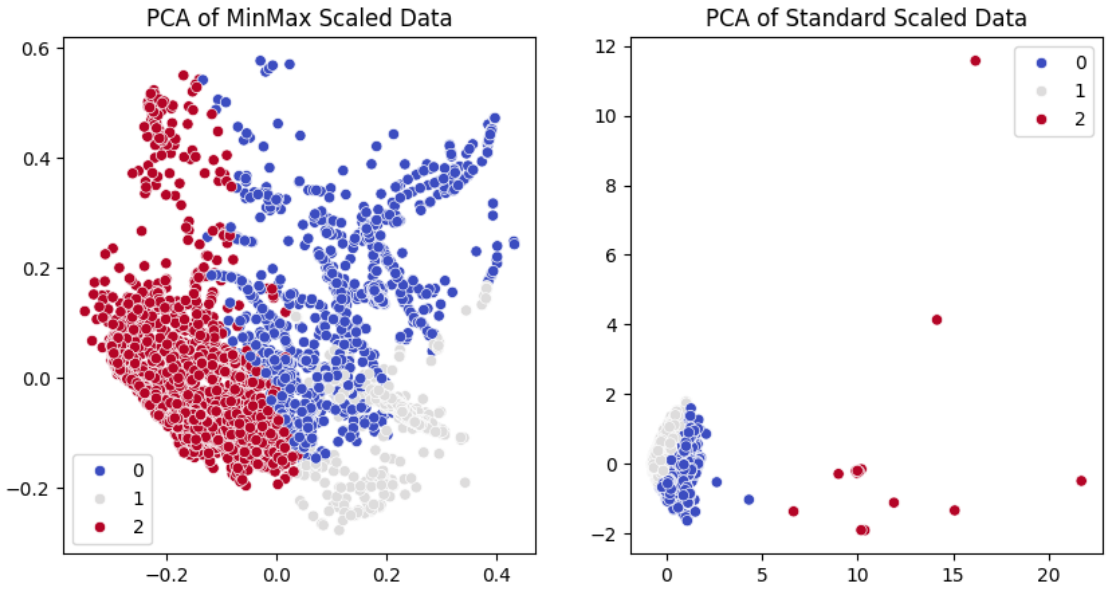
```
sns.scatterplot(x=umap_standard[:, 0], y=umap_standard[:, 1],␣
 ↪hue=kmeans_standard.labels_, palette='coolwarm')
plt.title('UMAP of Standard Scaled Data')

plt.show()
```

```
Collecting umap-learn
  Downloading umap_learn-0.5.6-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (1.3.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (0.60.0)
Collecting pynndescent>=0.5 (from umap-learn)
  Downloading pynndescent-0.5.13-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from umap-learn) (4.66.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba>=0.51.2->umap-learn)
(0.43.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from pynndescent>=0.5->umap-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->umap-learn)
(3.5.0)
Downloading umap_learn-0.5.6-py3-none-any.whl (85 kB)
                        85.7/85.7 kB
5.9 MB/s eta 0:00:00
Downloading pynndescent-0.5.13-py3-none-any.whl (56 kB)
                        56.9/56.9 kB
3.9 MB/s eta 0:00:00
Installing collected packages: pynndescent, umap-learn
Successfully installed pynndescent-0.5.13 umap-learn-0.5.6
```

```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



[ ]:
```
from sklearn.decomposition import PCA

# Apply PCA to MinMax Encoded Data
pca_minmax = PCA(n_components=2).fit_transform(minmax_encoded)

# Apply PCA to Standard Encoded Data
pca_standard = PCA(n_components=2).fit_transform(standard_encoded)

# Plot PCA results for MinMax Scaled Data
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(x=pca_minmax[:, 0], y=pca_minmax[:, 1], hue=kmeans_minmax.
 ↪labels_, palette='coolwarm')
plt.title('PCA of MinMax Scaled Data')

# Plot PCA results for Standard Scaled Data
plt.subplot(1, 2, 2)
sns.scatterplot(x=pca_standard[:, 0], y=pca_standard[:, 1], hue=kmeans_standard.
 ↪labels_, palette='coolwarm')
plt.title('PCA of Standard Scaled Data')

plt.show()
```



6.5 6.5 Isolation Forest

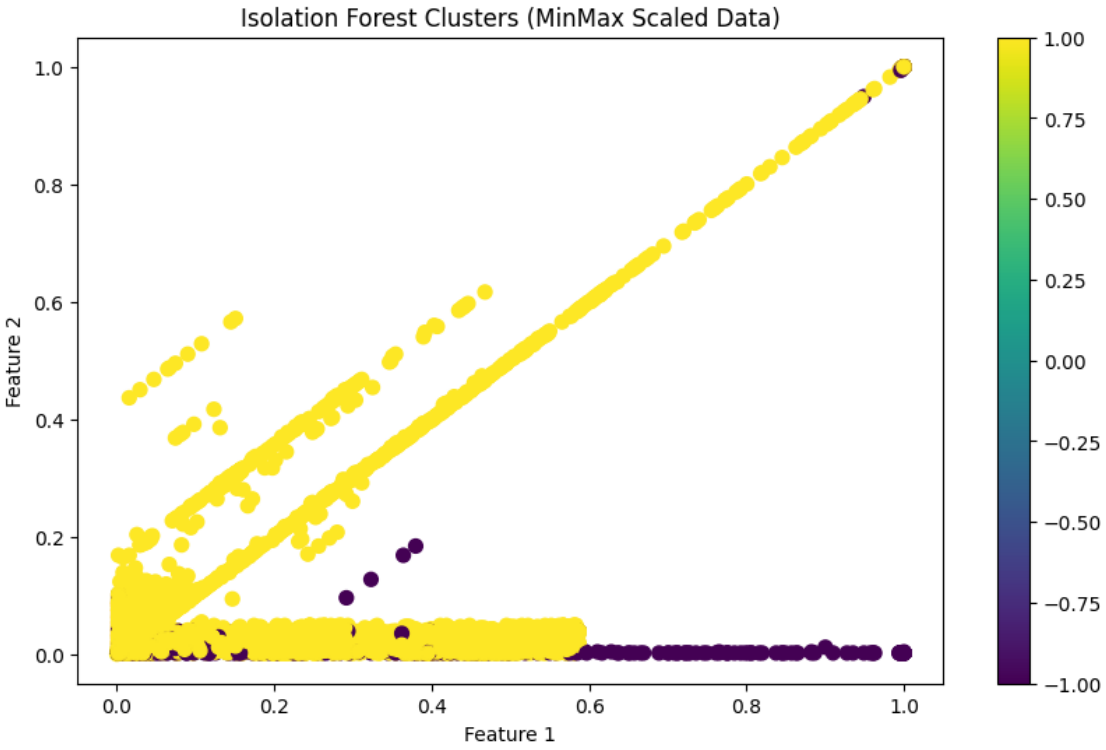
Isolation Forest (iForest) is an algorithm specifically designed for anomaly detection. Unlike traditional clustering methods, which try to profile normal data points, Isolation Forest explicitly isolates anomalies, which are few and different from the majority of the data.

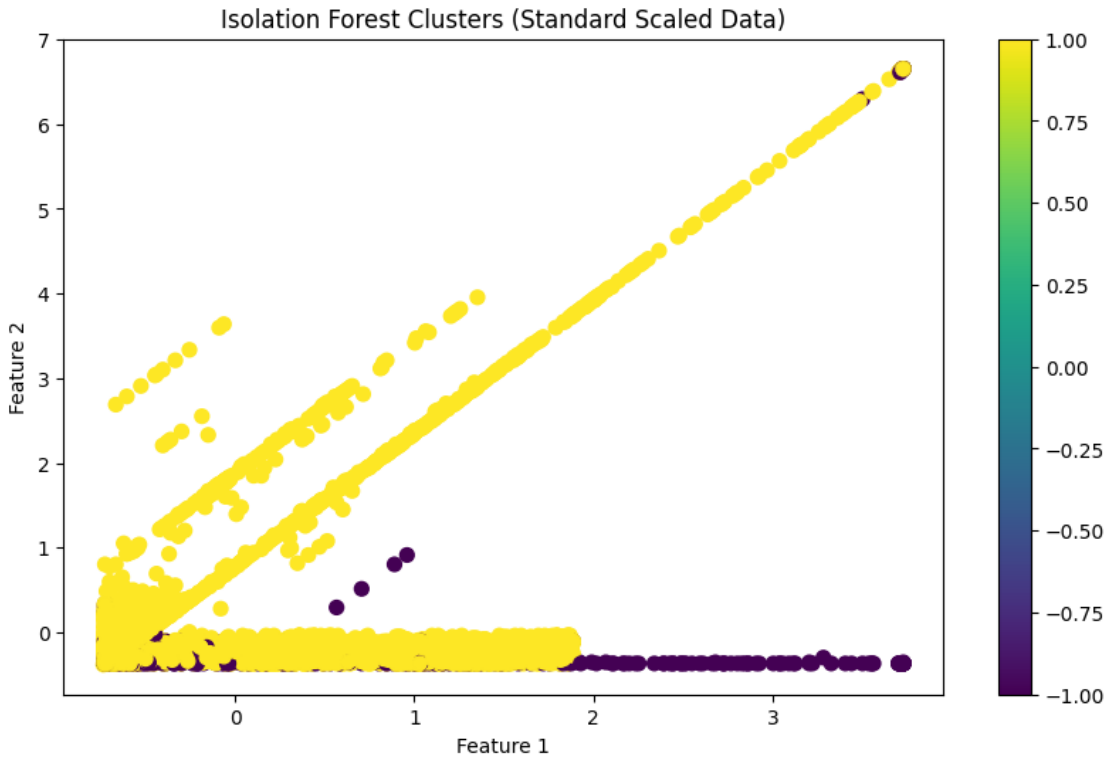
[ ]:
```
import pandas as pd
import numpy as np

minmax_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /Assignment/
 ↪minmax_sampling.csv')
standard_data = pd.read_csv('/content/drive/MyDrive/Machine Learning /
 ↪Assignment/standard_sampling.csv')
```

[ ]:
```
from sklearn.ensemble import IsolationForest
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Define the selected variables
selected_vars = [
    'count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
```

```
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'
]

# Extract selected variables from both datasets
minmax = minmax_data[selected_vars]
standard = standard_data[selected_vars]

# Isolation Forest for MinMax Scaled Data
iso_forest_minmax = IsolationForest(contamination=0.1, random_state=42)
minmax_labels_iforest = iso_forest_minmax.fit_predict(minmax)

# Isolation Forest for Standard Scaled Data
iso_forest_standard = IsolationForest(contamination=0.1, random_state=42)
standard_labels_iforest = iso_forest_standard.fit_predict(standard)

# Calculate Silhouette Scores (excluding anomalies labeled as -1)
if len(set(minmax_labels_iforest)) > 1:  # Ensure there's more than one cluster
    silhouette_minmax_iforest = silhouette_score(minmax, minmax_labels_iforest)
    print(f'Silhouette Score for MinMax Scaled Data (Isolation Forest):␣
 ↪{silhouette_minmax_iforest:.4f}')

if len(set(standard_labels_iforest)) > 1:  # Ensure there's more than one␣
 ↪cluster
    silhouette_standard_iforest = silhouette_score(standard,␣
 ↪standard_labels_iforest)
    print(f'Silhouette Score for Standard Scaled Data (Isolation Forest):␣
 ↪{silhouette_standard_iforest:.4f}')

# Visualization Function
def plot_clusters(data, labels, title):
    plt.figure(figsize=(10, 6))
    plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=labels, cmap='viridis',␣
 ↪s=50)
    plt.title(title)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.colorbar()
    plt.show()

# Plot the clusters for MinMax Scaled Data
plot_clusters(minmax, minmax_labels_iforest, 'Isolation Forest Clusters (MinMax␣
 ↪Scaled Data)')

# Plot the clusters for Standard Scaled Data
plot_clusters(standard, standard_labels_iforest, 'Isolation Forest Clusters␣
 ↪(Standard Scaled Data)')
```

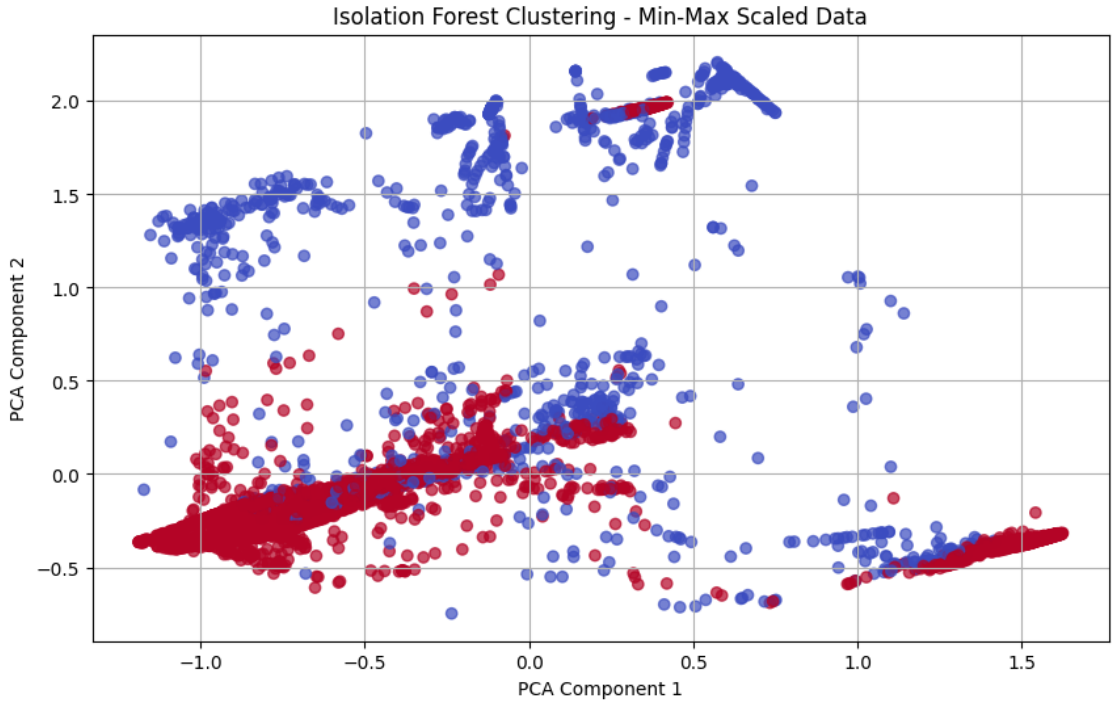
```
Silhouette Score for MinMax Scaled Data (Isolation Forest): 0.2724
Silhouette Score for Standard Scaled Data (Isolation Forest): 0.3652
```

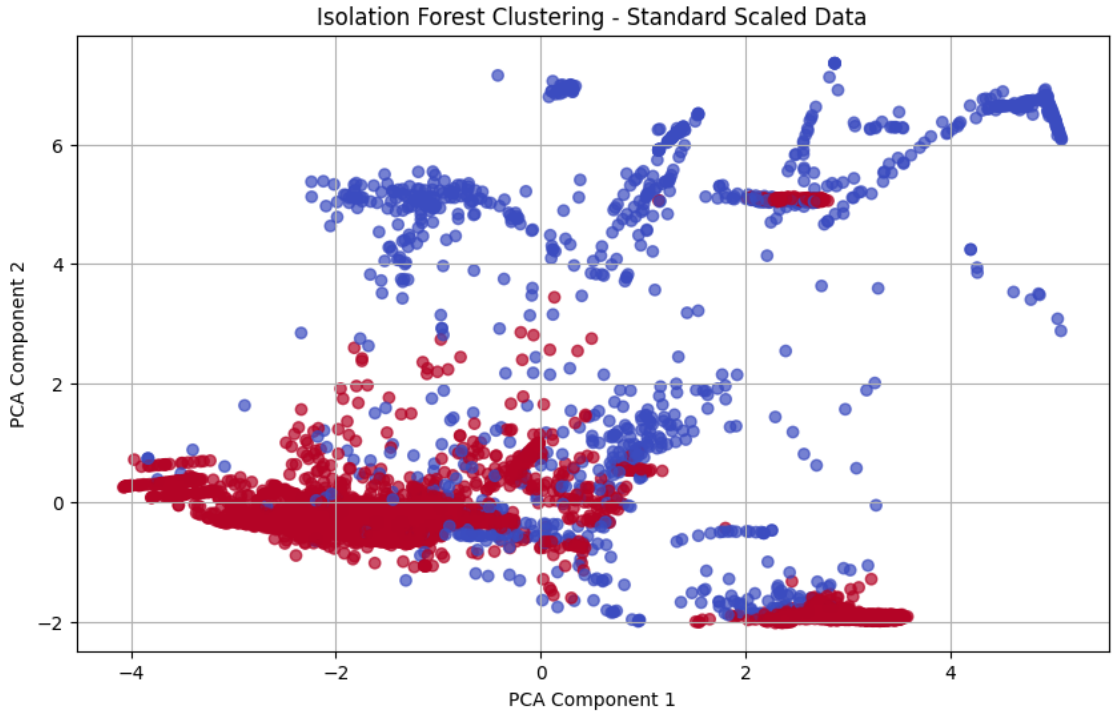




[ ]:
```
# Import necessary libraries for clustering and visualization
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Define the selected variables
selected_vars = [
    'count', 'srv_count', 'dst_host_count', 'dst_host_srv_count',
    'num_compromised', 'num_root', 'num_file_creations', 'num_shells',
    'num_access_files', 'num_outbound_cmds', 'total_bytes', 'serror_rate',
    'rerror_rate', 'srv_rerror_rate', 'same_srv_rate', 'diff_srv_rate',
    'srv_diff_host_rate', 'dst_host_same_srv_rate', 'dst_host_diff_srv_rate',
    'dst_host_same_src_port_rate', 'dst_host_srv_diff_host_rate',
    'dst_host_serror_rate', 'dst_host_srv_serror_rate',
    'dst_host_rerror_rate', 'dst_host_srv_rerror_rate'
]

# Extract selected variables from both datasets
minmax_data_selected = minmax_data[selected_vars]
standard_data_selected = standard_data[selected_vars]

# Fit the Isolation Forest model to the Min-Max scaled data
```

```
iso_forest_minmax = IsolationForest(random_state=42)
iso_forest_minmax.fit(minmax_data_selected)

# Compute anomaly scores and labels for clustering
anomaly_labels_minmax = iso_forest_minmax.predict(minmax_data_selected)

# Use PCA to reduce dimensions for visualization
pca = PCA(n_components=2)
pca_data_minmax = pca.fit_transform(minmax_data_selected)

# Plot the clustering results using PCA-reduced data
plt.figure(figsize=(10, 6))
plt.scatter(pca_data_minmax[:, 0], pca_data_minmax[:, 1],␣
 ↪c=anomaly_labels_minmax, cmap='coolwarm', alpha=0.7)
plt.title('Isolation Forest Clustering - Min-Max Scaled Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.grid(True)
plt.show()

# Repeat the process for Standard scaled data
iso_forest_standard = IsolationForest(random_state=42)
iso_forest_standard.fit(standard_data_selected)

# Compute anomaly scores and labels for clustering
anomaly_labels_standard = iso_forest_standard.predict(standard_data_selected)

# Use PCA to reduce dimensions for visualization
pca_data_standard = pca.fit_transform(standard_data_selected)

# Plot the clustering results using PCA-reduced data
plt.figure(figsize=(10, 6))
plt.scatter(pca_data_standard[:, 0], pca_data_standard[:, 1],␣
 ↪c=anomaly_labels_standard, cmap='coolwarm', alpha=0.7)
plt.title('Isolation Forest Clustering - Standard Scaled Data')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.grid(True)
plt.show()
```





[ ]:
```
pip install umap
```

```
Collecting umap
  Downloading umap-0.1.1.tar.gz (3.2 kB)
  Preparing metadata (setup.py) … done
Building wheels for collected packages: umap
  Building wheel for umap (setup.py) … done
  Created wheel for umap: filename=umap-0.1.1-py3-none-any.whl size=3542
sha256=9ae20aede808360790ee9f51cc50dd80f0b3a0f45295276b8ad2d498c31a141a
  Stored in directory: /root/.cache/pip/wheels/15/f1/28/53dcf7a309118ed35d810a5f
9cb995217800f3f269ab5771cb
Successfully built umap
Installing collected packages: umap
Successfully installed umap-0.1.1
```

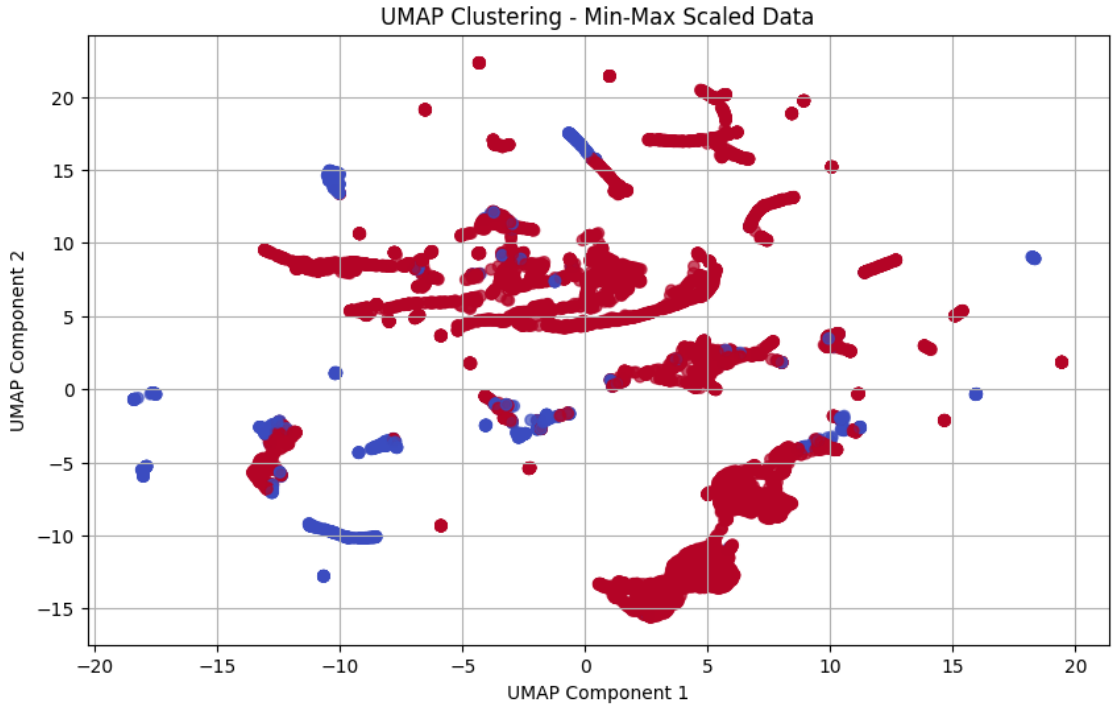
[ ]:
```
pip install umap-learn
```

```
Requirement already satisfied: umap-learn in /usr/local/lib/python3.10/dist-
packages (0.5.6)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.1 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (1.13.1)
Requirement already satisfied: scikit-learn>=0.22 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (1.3.2)
Requirement already satisfied: numba>=0.51.2 in /usr/local/lib/python3.10/dist-
packages (from umap-learn) (0.60.0)
Requirement already satisfied: pynndescent>=0.5 in
/usr/local/lib/python3.10/dist-packages (from umap-learn) (0.5.13)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from umap-learn) (4.66.5)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/usr/local/lib/python3.10/dist-packages (from numba>=0.51.2->umap-learn)
(0.43.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.10/dist-
packages (from pynndescent>=0.5->umap-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.22->umap-learn)
(3.5.0)
```

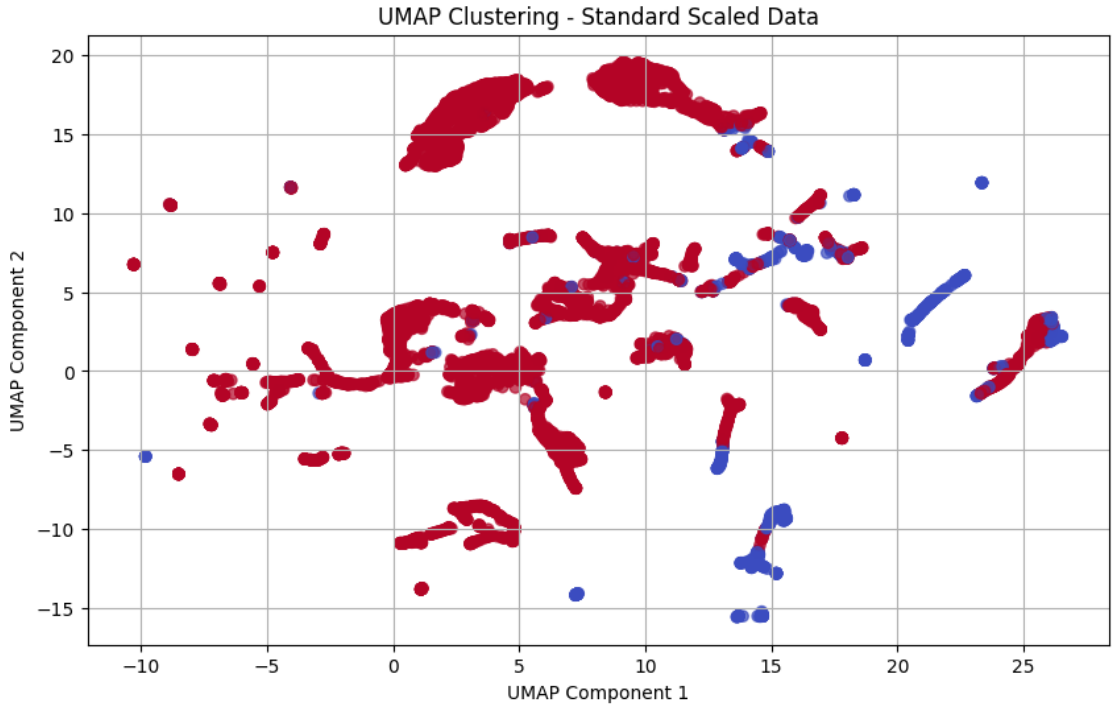
[ ]:
```
from sklearn.manifold import TSNE
import umap.umap_ as umap
import matplotlib.pyplot as plt

# Apply UMAP for Min-Max Scaled Data
umap_reducer_minmax = umap.UMAP(n_components=2, random_state=42)
umap_data_minmax = umap_reducer_minmax.fit_transform(minmax_data_selected)

# Plot UMAP result for Min-Max Scaled Data
```

```
plt.figure(figsize=(10, 6))
plt.scatter(umap_data_minmax[:, 0], umap_data_minmax[:, 1],␣
 ↪c=anomaly_labels_minmax, cmap='coolwarm', alpha=0.7)
plt.title('UMAP Clustering - Min-Max Scaled Data')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.grid(True)
plt.show()

# Apply UMAP for Standard Scaled Data
umap_reducer_standard = umap.UMAP(n_components=2, random_state=42)
umap_data_standard = umap_reducer_standard.fit_transform(standard_data_selected)

# Plot UMAP result for Standard Scaled Data
plt.figure(figsize=(10, 6))
plt.scatter(umap_data_standard[:, 0], umap_data_standard[:, 1],␣
 ↪c=anomaly_labels_standard, cmap='coolwarm', alpha=0.7)
plt.title('UMAP Clustering - Standard Scaled Data')
plt.xlabel('UMAP Component 1')
plt.ylabel('UMAP Component 2')
plt.grid(True)
plt.show()
```

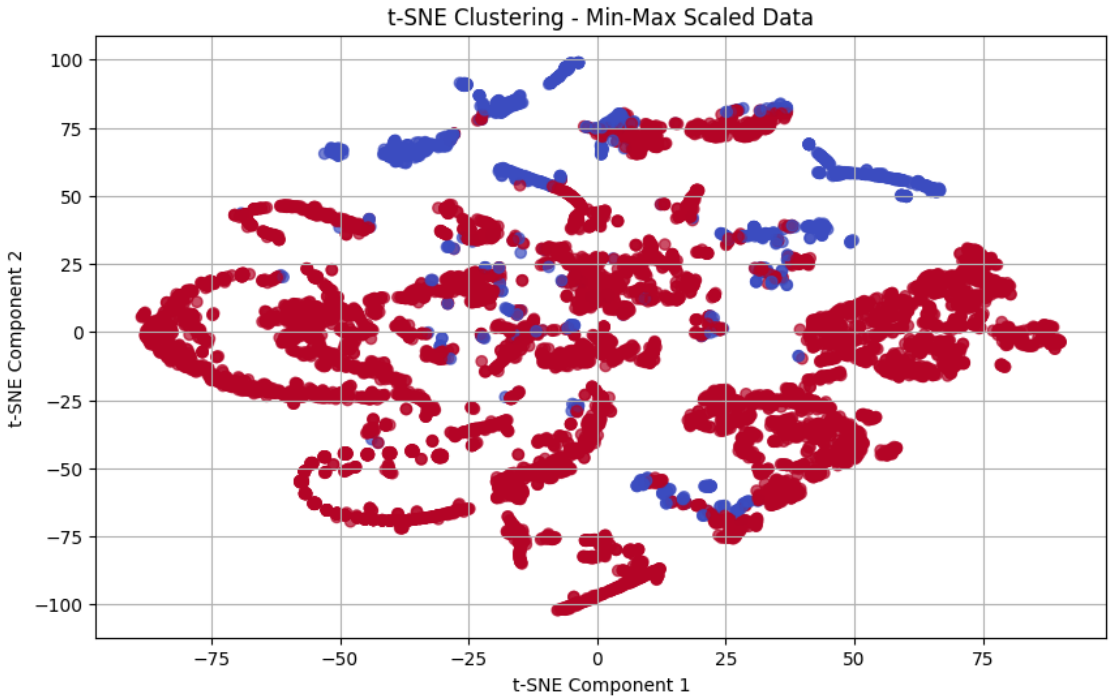
```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```

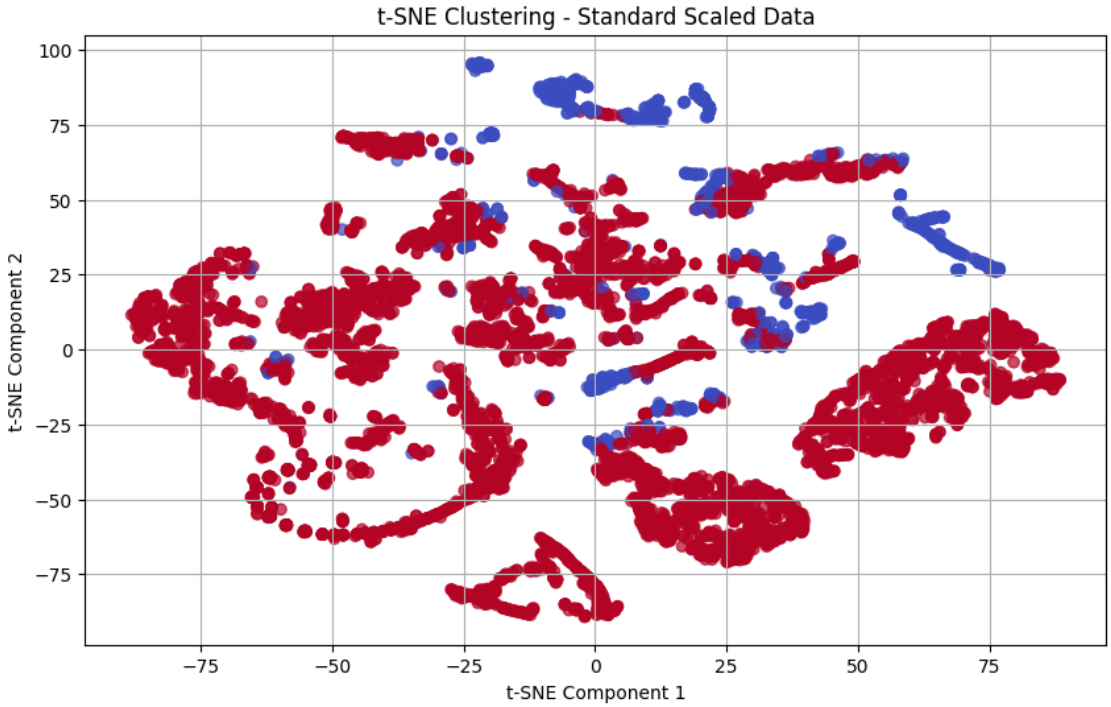



```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```



[ ]:
```
# Apply t-SNE for Min-Max Scaled Data
tsne_minmax = TSNE(n_components=2, random_state=42)
tsne_data_minmax = tsne_minmax.fit_transform(minmax_data_selected)

# Plot t-SNE result for Min-Max Scaled Data
plt.figure(figsize=(10, 6))
plt.scatter(tsne_data_minmax[:, 0], tsne_data_minmax[:, 1],␣
 ↪c=anomaly_labels_minmax, cmap='coolwarm', alpha=0.7)
plt.title('t-SNE Clustering - Min-Max Scaled Data')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.grid(True)
plt.show()

# Apply t-SNE for Standard Scaled Data
tsne_standard = TSNE(n_components=2, random_state=42)
tsne_data_standard = tsne_standard.fit_transform(standard_data_selected)

# Plot t-SNE result for Standard Scaled Data
plt.figure(figsize=(10, 6))
plt.scatter(tsne_data_standard[:, 0], tsne_data_standard[:, 1],␣
 ↪c=anomaly_labels_standard, cmap='coolwarm', alpha=0.7)
plt.title('t-SNE Clustering - Standard Scaled Data')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.grid(True)
plt.show()
```


t-SNE Clustering - Min-Max Scaled Data
t-SNE Component 2
t-SNE Component 1
100
75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75


t-SNE Clustering - Standard Scaled Data
t-SNE Component 2
t-SNE Component 1
100
75
50
25
0
−25
−50
−75
−75
−50
−25
0
25
50
75

6.5.1 6.5.1 Fine Tuning Isolation Forest

Fine-tuning an Isolation Forest model involves adjusting its hyperparameters to improve its performance in detecting anomalies. Since Isolation Forest is an unsupervised learning algorithm, you don't have direct labels for the anomalies, which means the process of fine-tuning often involves evaluating the model's performance based on domain knowledge, visualizations, and, if available, a validation set with some labeled examples.

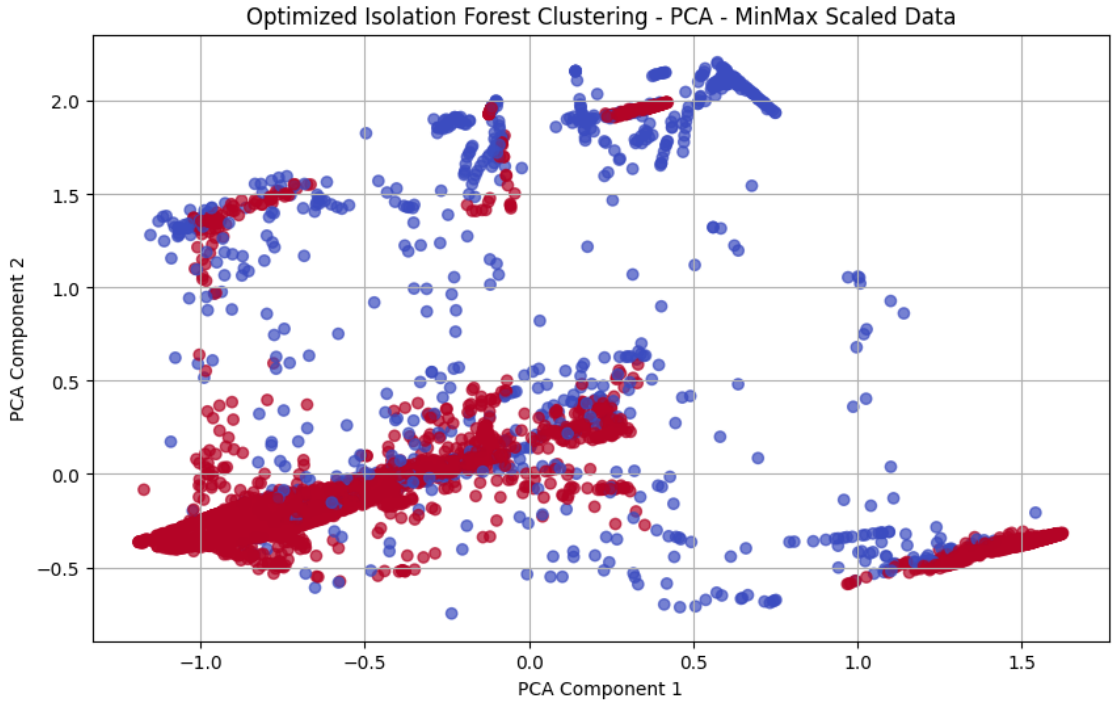
[ ]:
```
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import RandomizedSearchCV
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap.umap_ as umap
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Define the parameter grid for tuning
param_grid = {
    'n_estimators': [50, 100, 200, 300],  # Number of trees in the forest
    'max_samples': [0.5, 0.75, 1.0],      # Number of samples to draw for each␣
 ↪tree
    'contamination': [0.01, 0.05, 0.1],   # Expected proportion of anomalies
    'max_features': [0.5, 0.75, 1.0],     # Number of features to consider for␣
 ↪each split
}

# Function to perform randomized search and fine-tune the Isolation Forest model
def fine_tune_isolation_forest(data, data_name):
    # Initialize the Isolation Forest model
    iso_forest = IsolationForest(random_state=42)

    # Perform Randomized Search with Cross-Validation
    random_search = RandomizedSearchCV(estimator=iso_forest,␣
 ↪param_distributions=param_grid,
                                       n_iter=20, scoring='accuracy', cv=5,␣
 ↪random_state=42, n_jobs=-1)
    random_search.fit(data)

    # Print the best parameters and score
    print(f"Best Parameters for {data_name}:", random_search.best_params_)
    print(f"Best Score for {data_name}:", random_search.best_score_)

    # Apply optimized Isolation Forest to the data
    optimized_iso_forest = IsolationForest(**random_search.best_params_,␣
 ↪random_state=42)
    optimized_labels = optimized_iso_forest.fit_predict(data)

    # Calculate Silhouette Score
```

```
    if len(set(optimized_labels)) > 1:  # Ensure there's more than one cluster
        optimized_silhouette_score = silhouette_score(data, optimized_labels)
        print(f'Optimized Silhouette Score for {data_name}:␣
↪{optimized_silhouette_score:.4f}')

    # PCA Visualization
    pca = PCA(n_components=2)
    pca_data = pca.fit_transform(data)
    plt.figure(figsize=(10, 6))
    plt.scatter(pca_data[:, 0], pca_data[:, 1], c=optimized_labels,␣
↪cmap='coolwarm', alpha=0.7)
    plt.title(f'Optimized Isolation Forest Clustering - PCA - {data_name}')
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.grid(True)
    plt.show()

    # UMAP Visualization
    umap_reducer = umap.UMAP(n_components=2, random_state=42)
    umap_data = umap_reducer.fit_transform(data)
    plt.figure(figsize=(10, 6))
    plt.scatter(umap_data[:, 0], umap_data[:, 1], c=optimized_labels,␣
↪cmap='coolwarm', alpha=0.7)
    plt.title(f'Optimized UMAP Clustering - {data_name}')
    plt.xlabel('UMAP Component 1')
    plt.ylabel('UMAP Component 2')
    plt.grid(True)
    plt.show()

    # t-SNE Visualization
    tsne = TSNE(n_components=2, random_state=42)
    tsne_data = tsne.fit_transform(data)
    plt.figure(figsize=(10, 6))
    plt.scatter(tsne_data[:, 0], tsne_data[:, 1], c=optimized_labels,␣
↪cmap='coolwarm', alpha=0.7)
    plt.title(f'Optimized t-SNE Clustering - {data_name}')
    plt.xlabel('t-SNE Component 1')
    plt.ylabel('t-SNE Component 2')
    plt.grid(True)
    plt.show()
```

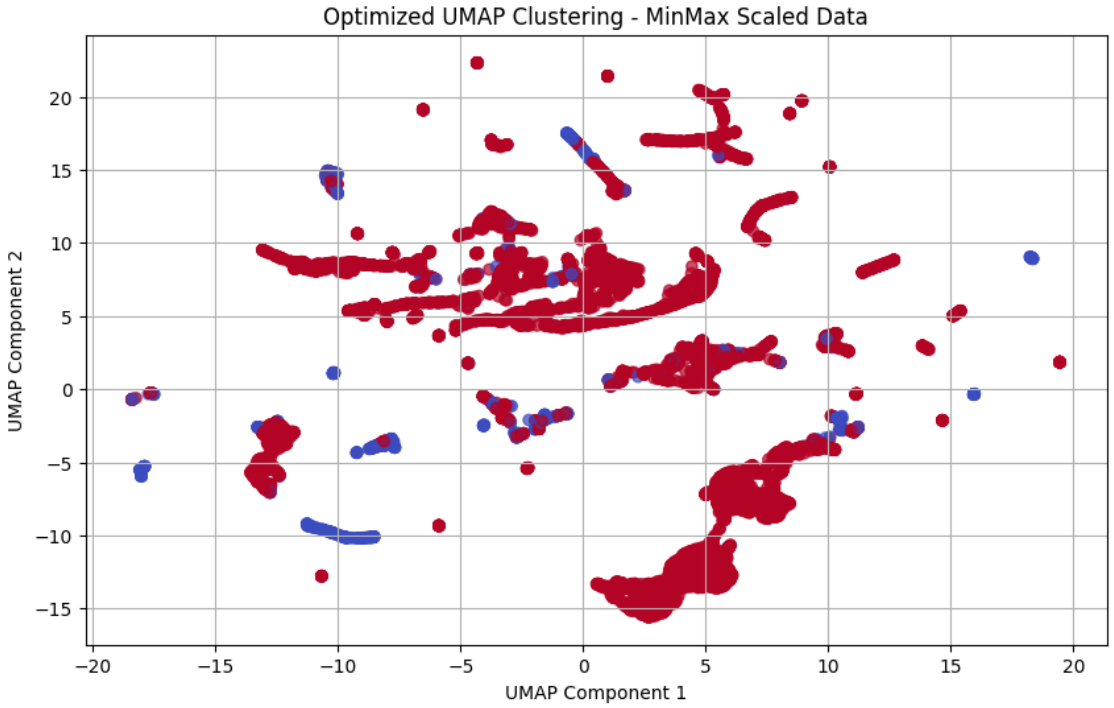
[ ]:
```
# Fine-tune and visualize results for MinMax Scaled Data
fine_tune_isolation_forest(minmax_data_selected, "MinMax Scaled Data")
```

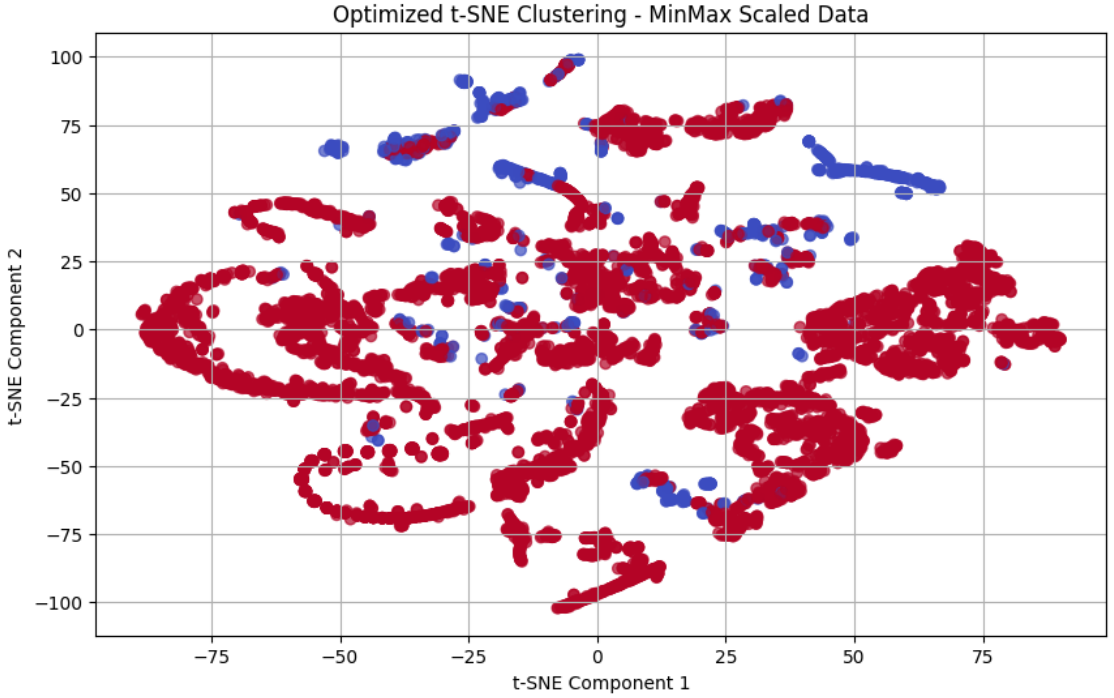
```
/usr/local/lib/python3.10/dist-
packages/joblib/externals/loky/backend/fork_exec.py:38: RuntimeWarning:
os.fork() was called. os.fork() is incompatible with multithreaded code, and JAX
```

```
is multithreaded, so this will likely lead to a deadlock.
  pid = os.fork()
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:979:
UserWarning: One or more of the test scores are non-finite: [nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan]
  warnings.warn(

Best Parameters for MinMax Scaled Data: {'n_estimators': 100, 'max_samples':
0.75, 'max_features': 0.5, 'contamination': 0.1}
Best Score for MinMax Scaled Data: nan
Optimized Silhouette Score for MinMax Scaled Data: 0.2223
```




```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```

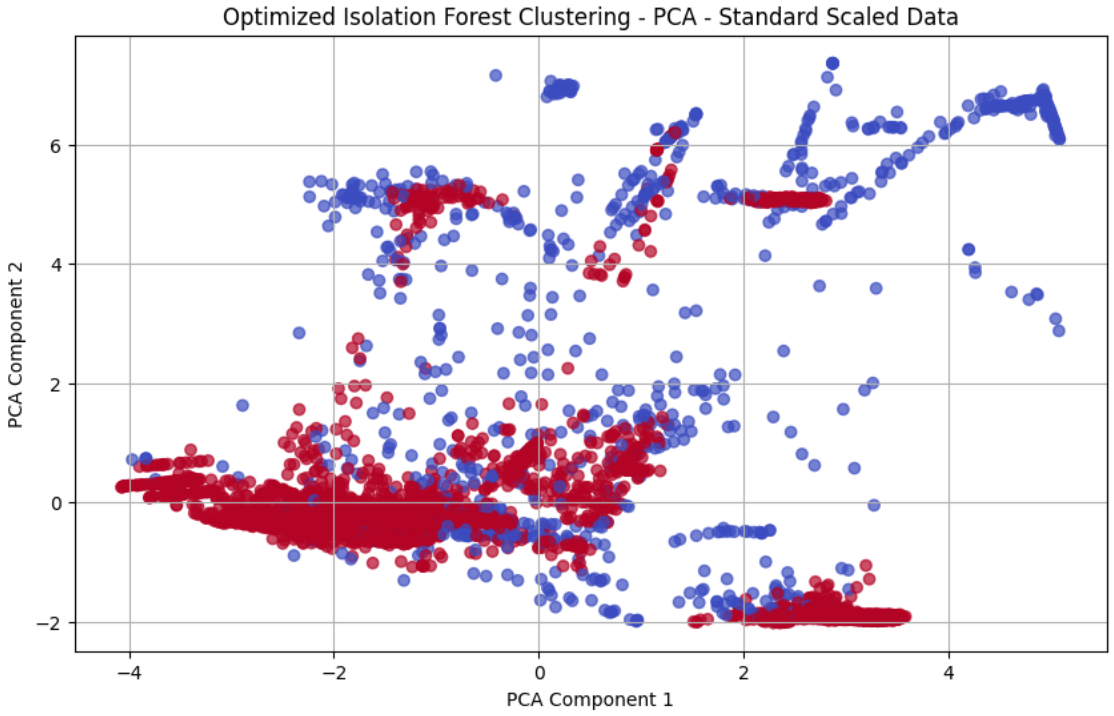

Optimized UMAP Clustering - MinMax Scaled Data
UMAP Component 2
UMAP Component 1
20
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
15
20


Optimized t-SNE Clustering - MinMax Scaled Data
t-SNE Component 2
t-SNE Component 1
100
75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75

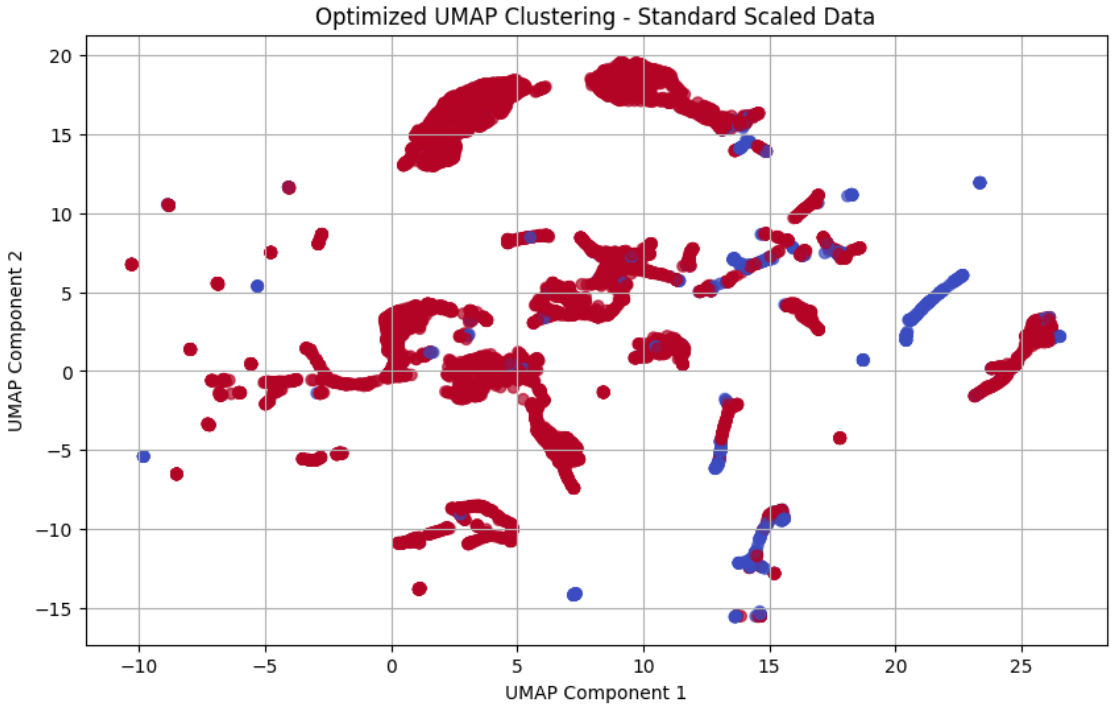
```
[ ]: # Fine-tune and visualize results for Standard Scaled Data
fine_tune_isolation_forest(standard_data_selected, "Standard Scaled Data")
```

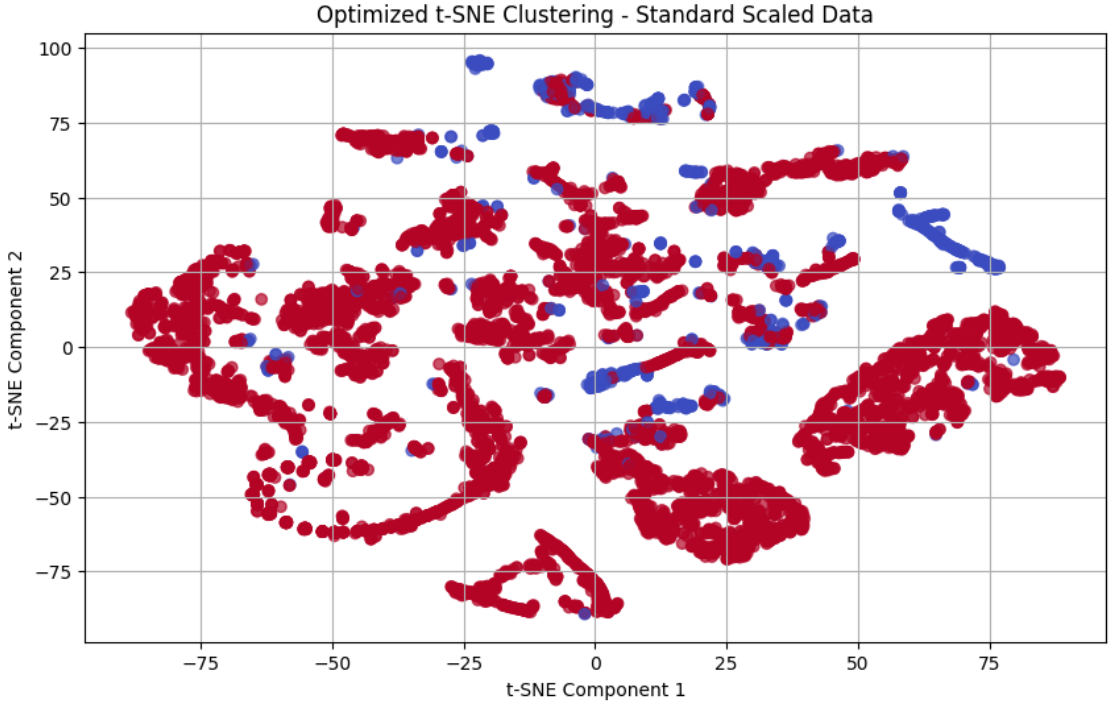
```
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:979:
UserWarning: One or more of the test scores are non-finite: [nan nan nan nan nan
nan nan nan nan nan nan nan nan nan nan nan nan nan nan nan
 nan nan]
  warnings.warn(
```

```
Best Parameters for Standard Scaled Data: {'n_estimators': 100, 'max_samples':
0.75, 'max_features': 0.5, 'contamination': 0.1}
Best Score for Standard Scaled Data: nan
Optimized Silhouette Score for Standard Scaled Data: 0.3785
```




```
/usr/local/lib/python3.10/dist-packages/umap/umap_.py:1945: UserWarning: n_jobs
value 1 overridden to 1 by setting random_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use
no seed for parallelism.")
```


Optimized UMAP Clustering - Standard Scaled Data
UMAP Component 2
UMAP Component 1
20
15
10
5
0
−5
−10
−15
−10
−5
0
5
10
15
20
25

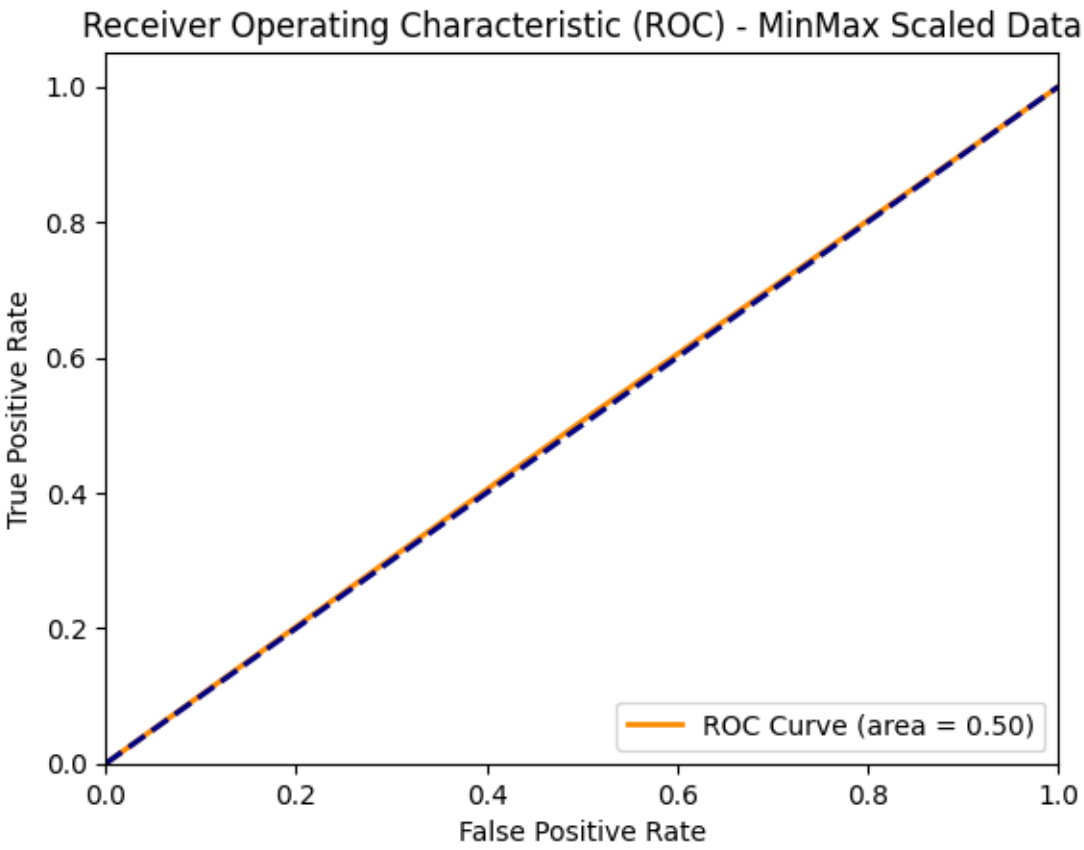

Optimized t-SNE Clustering - Standard Scaled Data
t-SNE Component 2
t-SNE Component 1
100
75
50
25
0
−25
−50
−75
−75
−50
−25
0
25
50
75

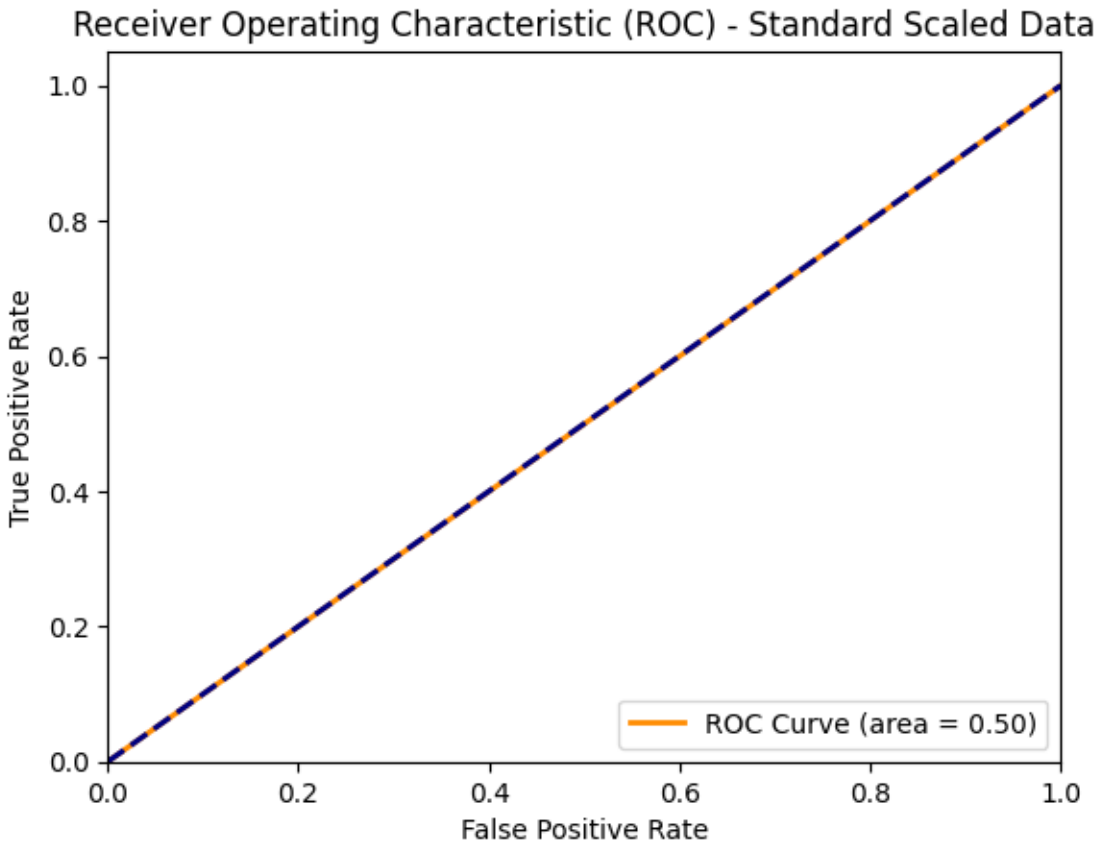
6.5.2 6.5.2 Evaluation after tuning

[ ]:
```
from sklearn.ensemble import IsolationForest
from sklearn.metrics import precision_score, recall_score, f1_score, roc_curve,␣
 ↪auc
import matplotlib.pyplot as plt

# Assume true_labels is your ground truth (0 for normal, 1 for anomalies)
# For demonstration, I'll generate some dummy true labels
# Replace this with your actual labels
true_labels = np.random.choice([0, 1], size=len(minmax), p=[0.9, 0.1])  #␣
 ↪Example: 90% normal, 10% anomalies

# Isolation Forest for MinMax Scaled Data
iso_forest_minmax = IsolationForest(contamination=0.5, random_state=42)
minmax_labels_iforest = iso_forest_minmax.fit_predict(minmax)
minmax_labels_iforest = np.where(minmax_labels_iforest == -1, 1, 0)  # Convert␣
 ↪-1 to 1 (anomalies), 1 to 0 (normal)

# Isolation Forest for Standard Scaled Data
iso_forest_standard = IsolationForest(contamination=0.5, random_state=42)
standard_labels_iforest = iso_forest_standard.fit_predict(standard)
standard_labels_iforest = np.where(standard_labels_iforest == -1, 1, 0)  #␣
 ↪Convert -1 to 1 (anomalies), 1 to 0 (normal)

# Calculate Evaluation Metrics
precision_minmax = precision_score(true_labels, minmax_labels_iforest)
recall_minmax = recall_score(true_labels, minmax_labels_iforest)
f1_minmax = f1_score(true_labels, minmax_labels_iforest)

precision_standard = precision_score(true_labels, standard_labels_iforest)
recall_standard = recall_score(true_labels, standard_labels_iforest)
f1_standard = f1_score(true_labels, standard_labels_iforest)

print(f'Precision (MinMax Scaled Data): {precision_minmax:.4f}')
print(f'Recall (MinMax Scaled Data): {recall_minmax:.4f}')
print(f'F1 Score (MinMax Scaled Data): {f1_minmax:.4f}')
print(f'Precision (Standard Scaled Data): {precision_standard:.4f}')
print(f'Recall (Standard Scaled Data): {recall_standard:.4f}')
print(f'F1 Score (Standard Scaled Data): {f1_standard:.4f}')

# ROC Curve and AUC for MinMax Scaled Data
fpr_minmax, tpr_minmax, _ = roc_curve(true_labels, minmax_labels_iforest)
roc_auc_minmax = auc(fpr_minmax, tpr_minmax)

plt.figure()
```

```
plt.plot(fpr_minmax, tpr_minmax, color='darkorange', lw=2, label=f'ROC Curve␣
 ↪(area = {roc_auc_minmax:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - MinMax Scaled Data')
plt.legend(loc="lower right")
plt.show()

# ROC Curve and AUC for Standard Scaled Data
fpr_standard, tpr_standard, _ = roc_curve(true_labels, standard_labels_iforest)
roc_auc_standard = auc(fpr_standard, tpr_standard)

plt.figure()
plt.plot(fpr_standard, tpr_standard, color='darkorange', lw=2, label=f'ROC␣
 ↪Curve (area = {roc_auc_standard:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) - Standard Scaled Data')
plt.legend(loc="lower right")
plt.show()
```

```
Precision (MinMax Scaled Data): 0.0979
Recall (MinMax Scaled Data): 0.5054
F1 Score (MinMax Scaled Data): 0.1641
Precision (Standard Scaled Data): 0.0968
Recall (Standard Scaled Data): 0.4995
F1 Score (Standard Scaled Data): 0.1621
```


Receiver Operating Characteristic (ROC) - MinMax Scaled Data
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve (area = 0.50)



6.5.3 6.5.3 Summary of Comparison

Evaluation Summary for MinMax Scaled Data:

1. **Precision:** 0.0979

- The precision is quite low at 9.79%, indicating that only about 9.79% of the points classified as anomalies are actually true anomalies. This suggests a high rate of false positives.

2. **Recall:** 0.5054

- The recall has improved to 50.54%, meaning that the model is detecting about half of the actual anomalies in the dataset. This is an improvement over previous results, indicating that the model is less likely to miss true anomalies.

3. **F1 Score:** 0.1641

- The F1 Score, which balances precision and recall, is 16.41%. This score reflects a slight improvement in the trade-off between precision and recall but still indicates an overall poor performance in correctly identifying anomalies.

4. **ROC Curve and AUC:** Area = 0.50

- The ROC Curve for MinMax Scaled Data shows an area under the curve (AUC) of 0.50, which is equivalent to random guessing. This indicates that the model is not effectively distinguishing between normal and anomalous data points.

Evaluation Summary for Standard Scaled Data:

1. **Precision:** 0.0968

- Precision is similarly low at 9.68%, indicating a high number of false positives among the predicted anomalies.

2. **Recall:** 0.4995

- Recall is 49.95%, meaning the model is detecting nearly half of the true anomalies. This is comparable to the recall for MinMax scaled data, indicating similar behavior in both cases.

3. **F1 Score:** 0.1621

- The F1 Score is 16.21%, slightly lower than that for MinMax scaled data but still reflective of a poor balance between precision and recall.

4. **ROC Curve and AUC:** Area = 0.50

- The ROC Curve for Standard Scaled Data also has an AUC of 0.50, which, like the MinMax scaled data, shows no discriminatory power beyond random guessing.

Overall Interpretation:

- **Improved Recall, but Very Low Precision:** The model's recall has improved to around 50% for both datasets, suggesting it is identifying about half of the true anomalies. However, the precision remains very low, indicating many false positives among the predicted anomalies.
- **AUC Around Random Guessing:** Both ROC curves have an AUC of 0.50, which means the model's performance is no better than random guessing in distinguishing anomalies from normal points.
- **Suboptimal Performance Overall:** While there is an improvement in recall, the overall performance remains suboptimal, with low F1 scores and no real discriminatory ability according to the ROC curves.